上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

THESIS OF BACHELOR

论文题目：语音识别任务中深度神经网络的结构优化

学生姓名：贺天行

学生学号：5100309127

专　　业：计算机科学与技术

指导教师：钱彦旻

学院(系)：电子信息与电气工程

语音识别任务中深度神经网络的结构优化

摘要

最近，由于算法和硬件方面的发展，深度神经网络（DNN）代替混合高斯模型（GMM）得以作为以隐马尔可夫模型（HMM）为基础的声学模型的一部分被引入到了自动语音识别系统中。无论在小规模数据集还是大词汇连续语音识别的任务上，DNN-HMM 框架的识别表现都大幅超越了经典的 GMM-HMM 框架，展现了 DNN 的强大建模能力。

虽然 DNN 模型在语音领域获得了巨大成功，但由于一个标准 DNN 模型包含了巨大数量的参数，使得用 DNN 模型进行解码十分耗时。已经有如删边（Weight truncation），矩阵分解等技术被提出来加速 DNN 的计算，这些方法利用了 DNN 模型的稀疏性。本文将总结以往重构 DNN 的工作，并提出一种“点剪枝”（Node-pruning）的方式来重构深度神经网络，以此降低模型复杂度。

本文提出的方法可以概括如下：首先得到一个 DNN 基线，然后使用某个准则将基线模型中的一些隐层神经元删去，然后重新进行训练。这样得到的模型可以直接用于解码，无需进行代码改动。更重要的是，点层面的重构可以与矩阵层面的重构相结合。在 Switboard 数据集上的实验中，点剪枝方法将基线模型的复杂度降到了 37.9%，在与奇异值矩阵分解（SVD）结合使用后，模型复杂度被降低至了 12.3%。后期实验也证明了 Node-pruning 能被用于加速 DNN 训练。

在这篇论文中，我们将首先介绍自动语音识别的关键概念和经典的 GMM-HMM 框架。接着将详细描述有关 DNN 训练的算法和流程。然后将会回顾已有的有关重构 DNN 模型的工作，并描述本文提出的 Node-pruning 算法流程。最后，将会展示和讨论实验结果。

关键词：深度神经网络，点剪枝，奇异值矩阵分解，大词汇连续语音识别

Structure Optimization of Deep Neural Network in Speech Recognition

ABSTRACT

Recent development of algorithms and hardware has enable researchers to introduce Deep neural network(DNN) as part of the Hidden Markov Model(HMM) based acoustic model for automatic speech recognition, replacing the Gaussian Mixture Model(GMM). The DNN-HMM framework outperforms the classic GMM-HMM framework from small-scale tasks to large vocabulary continuous speech recognition tasks, showing DNN's powerful modelling power.

Despite the huge success of the DNN-HMM framework, it is computationally expensive to deploy large-scale DNN in decoding due to huge number of parameters. Weights truncation and decomposition methods have been proposed to speed up decoding by exploiting the sparseness of DNN. This thesis will summarize different approaches of restructuring DNN and propose a new node-pruning approach to reshape DNN for reducing model complexity.

In this approach, hidden nodes of a fully trained DNN are pruned with certain importance function and the reshaped DNN is retuned using back-propagation. The approach requires no modification on code and can directly save computational costs during decoding. Furthermore, it is complementary to weight decomposition methods. Experiments on a switchboard task shows that, by using the proposed node-pruning approach, the complexity of a fully-trained DNN can be reduced to 37.9%. The complexity can be

further reduced to 12.3% without accuracy loss when node-pruning is combined with the (Singular Value Decomposition)SVD decomposition technique. We also show that node-pruning can be used to speed up DNN training.

This thesis first go over the key concepts in automatic speech recognition and the classic GMM-HMM framework. Then algorithms and procedure of the DNN-HMM ASR framework will be explained in detail. After that, related works about restructuring DNN will be reviewed, followed by the proposed node-pruning DNN training algorithm. Finally, experimental results will be shown and discussed.

Keywords: Deep Neural Networks, Node Pruning, Singular Value Decomposition, Large Vocabulary Continuous Speech Recognition

Contents

Chapter 1 Introduction

1.1 Automatic speech recognition

Speech is one of the most important ways of human communication. Researchers have worked for using machines to process speech since the invention of telephone in the late 19th century. Among speech process problems, automatic speech recognition(ASR), the task of converting recorded speech waveforms automatically into text, is one of the most challenging and most attractive tasks. Research on ASR progressed slowly in the early 20th century. In 1969, John Pierce of Bell Labs claimed that automatic speech recognition would not be a reality for several decades.

However, the 1970s witnessed a significant breakthrough in speech recognition invoked by the introduction of Hidden Markov model(HMM). In the following decades, HMMs were extensively investigated and became the most successful technique for acoustic modelling in speech recognition. The fast development of computer hardware and algorithms of discrete signal processing in the 1960s and 1970s also greatly stimulated interest in building ASR systems.

The aim of a speech recognition system is to produce a word sequence (or possibly character sequence for languages like Mandarin) given a speech waveform. The basic structure of an ASR system is shown in figure 1.1.

The first stage of speech recognition is to compress the speech signals to streams(frame by frame) of acoustic feature vectors, referred as observations. The extracted observation vectors are designed to contain sufficient information and be compact enough for performing efficient speech recognition. That process is called *front-end processing* or *feature extraction*. Given the observation feature vectors, the ASR system can be divided into three main components : the lexicon, language model and acoustic model. The lexicon, or dictionary, is used to map sub-word units in the acoustic model to actual words present in the vocabulary and language model. The language model represents the local syntactic and semantic information of the uttered sentences. It assigns

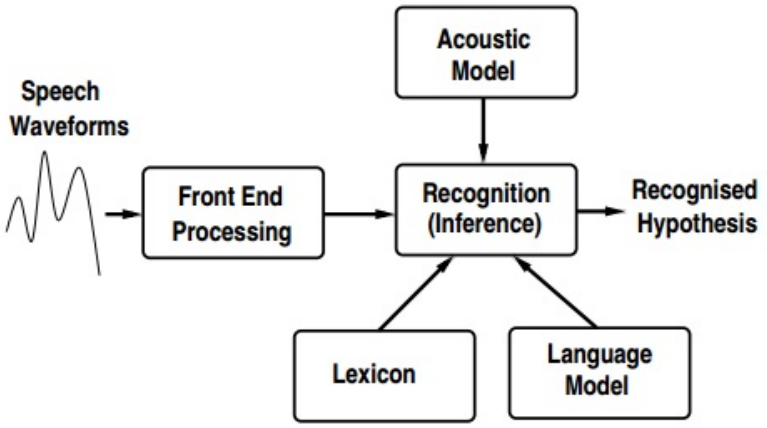

Figure 1.1 The general structure of an automatic speech recognition system

a possibility to each word sequences. The acoustic model, often of the most research interest in the speech community, maps the speech observation to sub-word units. In the GMM-HMM ASR framework, HMM is used to model the sub-word unit sequence of the given utterance, and on each HMM state, a GMM model is used to model the probability of the observation vector given the occurrence of that state.

Encouraged by its success, the GMM-HMM ASR system was soon applied to continuous speech recognition since it deals with isolated work recognition well. The vocabulary size also grows from about one thousand to tens of thousands of words, like the Wall Street Journal(WSJ) task. This kind of tasks are referred to as large vocabulary continuous speech recognition(LVCSR).

The GMM-HMM framework has been the state-of-art ASR framework until recently that Deep Neural Network(DNN) has been introduced into the ASR system. The proposed Deep-Neural-Network HMM (DNN-HMM) has achieved significant improvement over the state-of-art GMM-HMM for both phoneme recognition([1], rahman Mohamed et al., 2012: 14-22.) and large vocabulary continuous speech recognition(LVCSR) ([2], Dahl et al., 2012: 30-42.) tasks. It uses a deep neural network (DNN) to calculate posteriors of senones states and convert them into state-level conditional likelihood to be used in HMM.

Deep neural networks(DNN) have much more powerful modeling capacity than shallow ones, however, it is more likely trapped into a local optimum with the nor-

mal initialization strategy and back-propagation (BP) optimization. RBM (restricted Boltzmann machines) pre-training ([1], rahman Mohamed et al., 2012: 14-22.) or discriminative pre-training ([3], Bengio et al., 2007: 153-160.) are proposed to initialize the weights matrix to a better starting point, which make the BP finetune process more easily and convergence more quickly.

Recently more advanced training criteria are applied ([4], rahman Mohamed et al., 2010: 2846-2849.), ([5], Vesely et al., 2013: 2345-2349.) to get more refined model. Besides the hybrid framework, DNN could also be utilized to get more improved bottleneck features([6], Yu and Seltzer, 2011: 237-240.),([7], Sainath et al., 2012: 4153-4156.) which are used to train a GMM-HMM system and has achieved performance that matches DNN.

1.2 Main contribution of this thesis

To ensure good recognition performance, deep neural network (DNN) used in speech recognition, especially in LVCSR, usually has a *wide* and *deep* structure. For example, a typical CD-DNN-HMM with 3000 senone states as output has 7 hidden layers with about two thousand neurons per layer, leading to more than 30M parameters. The huge number of parameters incur significant computational costs (mainly due to large matrix multiplication) and memory requirements in decoding, which limits its application in real-world LVCSR([8], Vanhoucke et al., 2011: ?.). Hence, there has been great interest in studying how to reduce the DNN model complexity while preserving its powerful modeling capacity([9], Yu et al., 2012: 4409-4412.),([10], Xue et al., 2013: 2365-2369.),([11], Sainath et al., 2013: 6655-6659.), to speed up the decoding process.

Recent works all focused on exploiting the sparseness of weights in DNN, referred to as *weight operation* in this paper. One approach is direct *weight truncation*. It has been shown that by first throwing away the weights close to zero (up to 85% weights can be pruned) and then retuning the pruned network, DNN would not suffer any accuracy decline([9], Yu et al., 2012: 4409-4412.). However, it is non-trivial to use this approach to actually speed up decoding and special code implementation is required. Another type of weight operation is to restructure DNN by exploiting redundancy of weights

matrices calculation between layers. ([11], Sainath et al., 2013: 6655-6659.) aims to speed up both training and decoding by replacing the weight matrix below the output layer with the product of two small matrices. Further, ([10], Xue et al., 2013: 2365-2369.) applies singular value decomposition (SVD) on all weight matrices of a fully trained DNN, and retune the restructured model. These approaches can dramatically reduce DNN complexity while keeping the original recognition performance.

In contrast to the previous *weight operation* approaches, this paper focuses on *node* level restructuring of DNN. The proposed algorithm([12], Tianxing et al., 2014: 245-249.) reshapes a fully trained DNN by node-pruning, and then retunes it to prevent accuracy loss. Earlier works of node-pruning([13], Reed, 1993: 740-747.), ([14], Segee and J.Carter, 1991: 447-452.), ([15], Yamamoto et al., 1993: ?.) were all applied to small-scale shallow neural networks with the motivation of alleviating the over-fitting problem for better performance. Most of early proposed importance functions examine the outputs of a node. It has not been investigated that how node-pruning can help reduce complexity of a large scale DNN.

In this work, a new framework of node-pruning along with novel node importance functions are proposed for reshaping DNN. To the best of our knowledge, this is the first attempt to perform *node operation* in order to reduce complexity of DNN and apply it to LVCSR tasks. Considering that *node operation* is independent of weight operation, node-pruning can be combined with methods focusing on weights mentioned before to get additive improvements on model size reduction. To reduce model complexity, various number of hidden neurons are pruned by the node-pruning algorithm and test whether the pruned model has lost its original performance. Experiments on a 50 hour switchboard task shows that, by using the proposed node-pruning approach, DNN complexity can be reduced to 37.9%. The complexity can be further reduced to 12.3% without accuracy loss when node-pruning is combined with weight decomposition. We also show that node-pruning can be used to speed up DNN training.

1.3 Structure of this thesis

The rest of this thesis is organized as follows: In chapter 2 will review the classic GMM-HMM ASR framework, where HMM stands for Hidden Markov Model and GMM stands for Gaussian Mixture Model, we will discuss the components of the framework and how to build a standard large vocabulary continuous speech recognition(LVCSR) system. Chapter 3 talks about how to introduce the Deep Neural Network(DNN) into the ASR system, forming the DNN-HMM framework. In this chapter we describe the back-propagation(BP) algorithm and the Restricted Boltzmann Machine(RBM), which are used to train and pre-train a DNN, respectively. In chapter 4 we will first review some recent works about restructuring DNN with is mainly about exploiting the sparseness of DNN. Then we describe the node-pruning algorithm. Since the recent works do restructuring on the weight-level, and node-pruning is node-level, node-pruning can be complementary to any weight-level restructuring, in this work, we combine node-pruning with the SVD technique to further reduce model complexity. Finally in chapter 5 experimental results are shown.

Chapter 2 The GMM-HMM ASR framework

In this chapter will give an introduction to speech recognition systems using hidden Markov Models(HMMs) and Gaussian mixture models(GMM) as the acoustic model. Some material will also be given about front-end processing(feature extraction), language modelling and the search algorithm for decoding. The EM algorithm is used to train GMM-HMM model.

2.1 Front-end speech signal processing

The raw form of speech recorded is a continuous speech waveform. To effectively perform speech recognition, the speech waveform is normally converted into a sequence of time-discrete parametric vectors. These parametric vectors are assumed to give exact and compact representation of speech variabilities. These parametric vectors are often referred to as *feature vectors* or *observations*.

There are two widely used feature extraction schemes: Mel-frequency Cepstral coefficients (MFCC)([16], Davis and Mermelstein, 1980: 357-366.) and perceptual linear prediction (PLP)([17], H.Hermansky, 1990: 1738-1752.). Both schemes are based on Cepstral analysis. The initial frequency analysis of the two schemes are the same. First, the speech signal is split into discrete segments usually with 10ms shifting rate and 25ms window length. This reflects the short-term stationary property of speech signals([18], Rabiner and Juang, 1993: ?.). These discrete segments are often referred to as frames. A feature vector will be extracted for each frame. A pre-emphasising technique is normally used during the feature extraction, where overlapping window functions, such as Hamming or Hanning windows are used to smooth the signals. Using a window function reduces the boundary effect in signal processing. A fast Fourier transform (FFT) is then performed on the time-domain speech signals of each frame, generating the complex frequency domains. Having obtained the frequency domain for each frame, different procedures are used to obtain MFCC or PLP features. The

difference lies in the frequency warping methods and the Cepstral representation.

When using hidden Markov models (HMMs) as the acoustic model, which will be introduced in the next section, there is a fundamental assumption that the observations are conditionally independent. This requires removal of the temporal correlation of speech signals. Dynamic coefficients may be incorporated into the feature vector to reduce the temporal correlation. These dynamic coefficients represent the correlation between static feature vectors of different time instances. One common form is the delta coefficient , $\Delta \mathbf{o}_t$, which is calculated as a linear regression over a number of frames.

$$\Delta \mathbf{o}_t = \frac{\sum_{k=1}^{K} k(\mathbf{o}_{t+k} - \mathbf{o}_{t-k})}{2\sum_{k=1}^{K} k^2} \tag{2-1}$$

2.2 The GMM-HMM ASR framework

The task of ASR is that given a stream of observation $\mathbf{O} = [\mathbf{o}_1, ...\mathbf{o}_T]$, output the most likely word sequence $\mathcal{H}$, so

$$\hat{\mathcal{H}} = \arg\max_{\mathcal{H}} p(\mathbf{O}|\mathcal{H}) \tag{2-2}$$

Now by apply the *Bayes Rule*:

$$\begin{aligned} \hat{\mathcal{H}} &= \arg\max_{\mathcal{H}} \{\frac{p(\mathbf{O}|\mathcal{H})p(\mathcal{H})}{p(\mathbf{O})}\} \\ &= \arg\max_{\mathcal{H}} \{p(\mathbf{O}|\mathcal{H})p(\mathcal{H})\} \end{aligned} \tag{2-3}$$

Here, $p(\mathbf{O}|\mathcal{H})$ is modeled by acoustic model and $p(\mathcal{H})$ is modeled by language model. In this section we will first focus on acoustic modelling.

A hidden Markov model (HMM) is a statistical generative model. It is very popular and successful in speech recognition. In HMM based speech recognition, the speech observations of a particular acoustic unit, such as a word or a phone, are assumed to be generated by a finite state machine. The state changes at each time unit with a certain probability. At each time instance, when a state is entered, an observation is generated from some probability function.

2.2.1 Acoustic Units

For human beings the basic unit of speech are words, but for a realistic automatic speech recognition task, the vocabulary is usually very large(larger than 10k words).So for statistical machine learning, we don't have enough data to learn the model for each word. One widely used solution is to use HMM to model *sub-word* units. *Phone* is a widely used sub-word unit. The number of phones is normally significantly smaller than the size of vocabulary. For example, we can use 46 distinct phones to constitute words in a vocabulary whose size is 64K.

There are two main types of phone model sets used, mono-phones which are context-independent phones, and context-dependent phones. The mono-phone set uses each individual phone as the sub-word unit and does not take into account the context information. Due to the co-articulatory effect, the pronunciation of the current phone is highly dependent on the preceding and following phones. Thus, for many speech recognition tasks, the use of mono-phones do not yield good performance.

To model these variations, context dependent phones are used in most state-of-the-art speech recognition systems. One commonly used context-dependent phone set is the tri-phone, which takes into account the preceding and following phones of the current phone. For example, consider the phone ah, a possible tri-phone may be w-ah+n, where w is the preceding phone and n is the following phone. Therefore, an isolated word "one" can be mapped into the following tri-phone sequence:

one=sil-w+ah w-ah+n ah-n+sil

In the GMM-HMM framework, each tri-phone has a HMM to model, a *dictionary*, or a *lexicon* is needed to map words to tri-phones. Next we give definition to the HMM models.

2.2.2 Hidden Markov Models for acoustic modelling

A left-to-right HMM with three emitting states is shown in figure 2.1, this is a typical topology of HMM used in speech recognition, the number of states could vary.

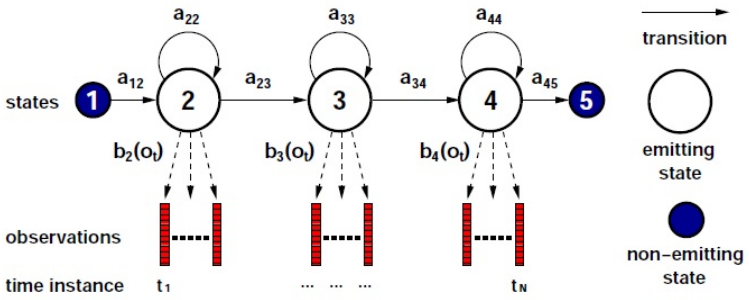

Figure 2.1 A left-to-right HMM with three emitting states

In HMM based speech recognition, the speech observations of a particular acoustic unit, such as a word or a phone, are assumed to be generated by a finite state machine. The state changes at each time unit with a certain probability. At each time instance, when a state is entered, an observation is generated from some probability function. The use of HMMs is dependent on a number of assumptions:

- **Quasi-stationary:** Speech signals may be split into short segments corresponding to the hidden states, in which the waveform is considered to be stationary and feature vectors can be extracted. The transitions between these states are assumed to be instantaneous.

- **Conditional independence:** Each observation is assumed to be generated with a certain probability associated with a hidden state. This means the observation is only dependent on the current state and is conditionally independent of both the previous and the following observations, given the state.

Neither of the two assumptions is true for real speech. Much research has been carried out to compensate the effect of the poor assumptions or to find alternative models for speech. However, the standard HMM is still a successful acoustic modelling technique and is widely used in most speech recognition systems.

Given the observation sequence $\mathbf{O} = [\mathbf{o}_1, ...\mathbf{o}_t]$ corresponding to a specific acoustic unit(a tri-phone) which is modelled by a HMM. These observations are assumed to be generated from the three emitting states of the HMM. The generation process starts from the first non-emitting state. At each time instance, the state transits with a certain

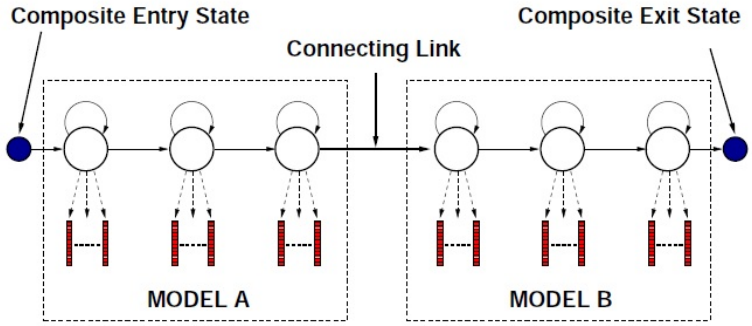

Figure 2.2 A composite HMM model

probability to either itself or the contiguous right state. So each edge e_{ij} is assigned a discrete probability a_{ij}. On each emitting state j, suppose the current observation is $\mathbf{o}_i$, then probability of the observation at that time instance generated by state j is modelled by a probability density $b_j(\mathbf{o}_t)$. Usually Gaussian Mixture Model is used for b_j, which we shall discuss later.

Note that the entry and exit status in figure 2.1 are non-emitting, they are used for construction of composite HMMs. Because for continuous speech recognition, we need to model a sequence of acoustic units, so we need to "glue" the hmms together to form a network. This is illustrated in figure 2.2.

As a summary, we now define the parameter set $\mathcal{M}$ for a HMM model:

- π-initial state distribution

  Initial state distribution is used to describe from which state does the process begin. Let w_t denote the state at time instance t, the initial state distribution is expressed as

  $$P(w_1 = i) = \pi_i \tag{2-4}$$

  In our case, the initial state distribution for the entry state is always 1.

- State transition matrix

  The element of state transition matrix A is defined as:

  $$a_{ij} = P(w_{t+1} = j | w_t = i) \tag{2-5}$$

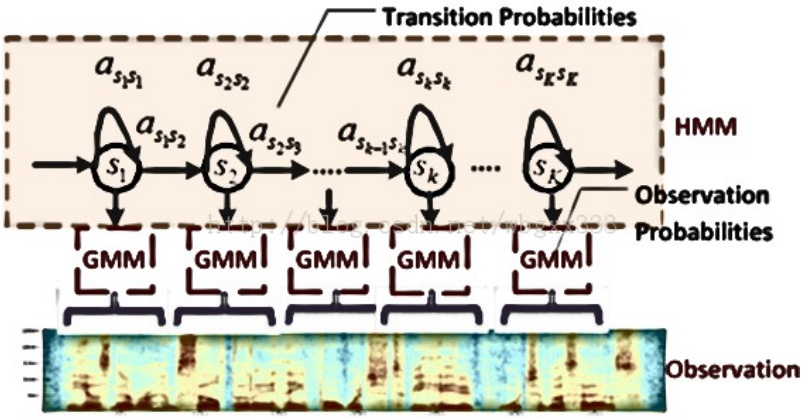

Figure 2.3 The GMM-HMM framework

- State output distributions

  The parameters for the output distributions of the emitting states need also be included, usually the multivariate Gaussian mixture model (GMM) is used, which is a linear superposition of k Guassians:

$$p(\mathbf{o}) = \Sigma_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{o}|\mu_k, \Sigma_k) \tag{2-6}$$

  subject to:

$$\Sigma_{k=1}^{K} \pi_k = 1 \tag{2-7}$$

  And $\mathcal{N}$ means the standard multivariate Gaussian.

  Each emitting state has a GMM model as the output density. the covariance matrix Σ is normally assumed to be diagonal. The use of a **diagonal** covariance matrix may give poor modelling of correlation between different dimensions. Hence, many complicated covariance modelling techniques have been investigated. However, diagonal covariance matrices are still widely used because of the low computational cost and their successful use in state-of-art art LVCSR systems.

The HMM-GMM framework is depicted in figure 2.3.

2.2.3 Likelihood calculation of GMM-HMM

The likelihood calculation $p(\mathbf{O}|\mathcal{M})$ for HMM is complicated, first, assuming the generation process goes through a state sequence $\mathbf{w} = w_1, ..., w_t$, the likelihood is then:

$$p(\mathbf{O}|\mathbf{w}, \mathcal{M}) = a_{w_1 w_2} \prod_{i=2}^{t-1} \{b_{w_i}(\mathbf{o}_i) a_{w_i w_{i+1}}\} \tag{2-8}$$

which is the product of its passing transition probability and emitting probability density. Now, for $p(\mathbf{O}|\mathcal{M})$ we need to consider all possible state sequence:

$$p(\mathbf{O}|\mathcal{M}) = \sum_{\mathbf{w}} p(\mathbf{O}|\mathbf{w}, \mathcal{M}) \tag{2-9}$$

The GMM-HMM is trained by maximum likelihood estimation, which maximizes the log-likelihood:

$$\hat{\mathcal{M}} = \arg\max_{\mathcal{M}} \{log(p(\mathbf{O}|\mathcal{M}))\} \tag{2-10}$$

Due to the existence of hidden variables $\mathbf{w}$ in HMMs, direct optimisation of equation 2-10 with respect to $\mathcal{M}$ is nontrivial. One solution for this type of optimisation is the expectation maximisation (EM) algorithm. One iteration of the EM algorithm consists of two steps:

- **Expectation:** Obtain the state posterior given the current model estimate to form an auxiliary function.
- **Maximisation:** Maximise the auxiliary function to get a new model estimate.

Since the training of HMM is not the focus of this thesis, we refer the readers to ([19], Yu, 2006: ?.) for details about the EM algorithm. Note that the EM algorithm requires an initialization, a flat estimate can be used to start the iterative EM algorithm.

2.2.4 State clustering

In section 2.2.1 we reviewed that for a standard LVCSR task we usually use triphones as acoustic units to capture context information. However, we will meet the

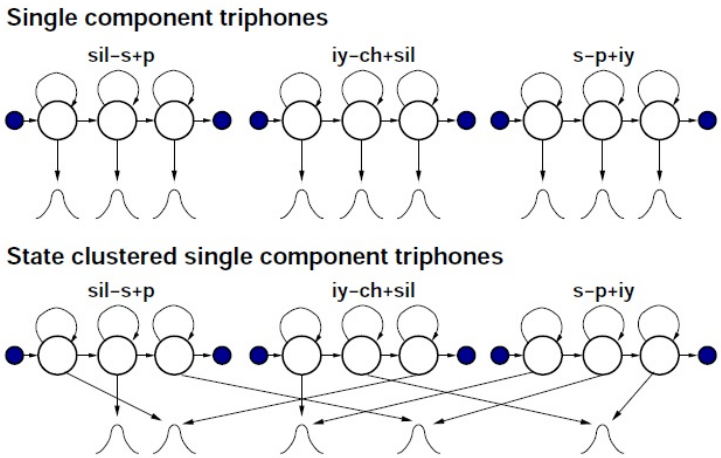

Figure 2.4 State clustering for single Gaussian tri-phones

same problem: some tri-phones don't have enough data to train. To solve this problem, parameter tying, or clustering techniques are often used. The basic idea of the technique is to consider a group of parameters as sharing the same set of values. In training, statistics of the whole group is used to estimate the shared parameter. Tying can be performed at various levels, such as phones, states, Gaussian components, or even mean vectors or covariance matrices of Gaussian components.

The most widely used approach is to do state level parameter tying, referred to as state clustering. In state clustering, an output distribution is shared among a group of states as illustrated in figure 2.4.

A widely used method is decision tree based state clustering. A phonetic decision tree is a binary tree with a set of yes or no questions about the left and right context of each phone. Clustering is performed top-down. At first all states are grouped at the root node, the states are then split into child nodes by answering some context questions. This split process stops when the amount of training data associated with the current node falls below a minimum threshold. The question selected at each split is the one that locally maximises the likelihood increase.

2.3 A standard training procedure for LVCSR system

In the last few sections we go over the basic concepts about GMM-HMM framework. Here, as a wrap-up, we list the key steps about building a HMM based ASR system below:

- **Build a monophone system:**
  - Train single Gaussian for each phone as flat start
  - Realign training transcriptions.
- **Build a cross-word triphone system:**
  - Clone mono-phone models to tri-phone models.
  - Train for one or two iterations using unclustered tri-phone models.
  - Perform decision tree state clustering.
  - Perform re-estimation.
  - Iterative Gaussian mixture splitting.
- **Rebuild decision tree:**
  - 2-model re-restimate single Gaussian unclustered triphone system using the final model alignment from last step.
  - Re-perform decision tree state clustering.
  - Perform re-estimation.
  - Iterative Gaussian mixture splitting.

2.4 Speech recognition using HMM

As we have discussed before, the task of ASR is that given a stream of observation $\mathbf{O} = [\mathbf{o}_1, ...\mathbf{o}_T]$, output the most likely word sequence $\mathcal{H}$, so

$$\hat{\mathcal{H}} = \arg\max_{\mathcal{H}} P(\mathbf{O}|\mathcal{H}) \tag{2-11}$$

Now by applying the *Bayes Rule*:

$$\begin{aligned}\hat{\mathcal{H}} &= \arg\max_{\mathcal{H}}\{\frac{p(\mathbf{O}|\mathcal{H})p(\mathcal{H})}{p(\mathbf{O})}\} \\ &= \arg\max_{\mathcal{H}}\{p(\mathbf{O}|\mathcal{H})p(\mathcal{H})\}\end{aligned} \tag{2-12}$$

It can be further written as

$$\begin{aligned}\hat{\mathcal{H}} &= \arg\max_{\mathcal{H}}\{log(p(\mathbf{O}|\mathcal{H})p(\mathcal{H}))\} \\ &= \arg\max_{\mathcal{H}}\{log(p(\mathbf{O}|\mathcal{H})) + log(p(\mathcal{H}))\}\end{aligned} \tag{2-13}$$

The first term is called the acoustic score for $\mathcal{H}$ and the second term is called the language score.

2.4.1 Language modelling

Language model is used to assign a probability to a word sentence $\mathcal{H}$. The probability of a sentence can be factorized into product of conditional probabilities:

$$P(\mathcal{H}) = \prod_{k=1}^{K} P(w_k|w_{k-1}, ..., w_1) \tag{2-14}$$

where w_k is the kth word.

For LVCSR, the word sequence is usually too long to be modelled properly. One solution is to restrict the length of history. One widely used language model using this strategy is the *N-gram*, which assumes:

$$P(w_k|w_{k-1}, ..., w_1) \approx P(w_k|w_{k-1}, ..., w_{k-N+1}) \tag{2-15}$$

N is usually very small, for example when $N = 3$ it's called *tri-gram*. Given a training text data, we can easily get MLE result for a tri-gram model:

$$P(w_k|w_{k-1}, w_{k-2}) = \frac{count(w_k, w_{k-1}, w_{k-2})}{\sum_w count(w, w_{k-1}, w_{k-2})} \tag{2-16}$$

where $count$ means the number of occurrence in the training data.

The N-gram model has a serious problem called data sparsity, for example, if a word tuple which didn't appear in the training data would receive zero probability. Therefore many smoothing techniques have been used for compensating, we refer readers to ([20], Chen and Goodman, 1996: 310-318.). Also note that recently neural network based language model has begun to show promising performance, but N-gram remains the most popular for its speed and simplicity.

2.4.2 Search and decoding

(2-9) showed that for calculating the probability of a word sequence, all possible state sequence **w** need to be considered. However, in actual decoding we only search for the best state sequence, which means we are assuming:

$$p(\mathbf{O}|\mathcal{H},\mathcal{M}) \approx max_{\mathbf{w}} p(\mathbf{O}|\mathbf{w},\mathcal{H},\mathcal{M}) \tag{2-17}$$

For searching the most likely state sequence, a *token passing* algorithm is used. The search process is frame by frame, on each time instance, each HMM state has some tokens landed on it. A token records the likelihood of the partial path and a pointer to the history of the HMM sequence. At each time instance, these tokens are updated and propagated forward and corresponding acoustic score or language score is added. There are many practical issues with the token-passing algorithm, for example, pruning is needed to prevent too many alive tokens. Finally, the token with the best score is traced back to give the most likely state sequence.

2.5 Summary

In this chapter, we introduce the key concepts and algorithm for building a standard HMM based lcvsr system. The ASR system is a highly comprehensive system involving many model components and parameters to learn or tune. The GMM-HMM framework has shown promising performance, and has been the state-of-art ASR system for decades. We have to note that many important training techniques like HLDA

or MPE training have not been discussed in this chapter.

Chapter 3 The DNN-HMM ASR framework

In the last chapter, we reviewed the GMM-HMM ASR framework, which has brought huge accuracy improvement to the ASR community, and it is so successful that it is difficult for new methods to outperform GMM-HMM for acoustic modeling. But GMMs have a serious shortcoming that they are statistically inefficient for modeling data that lie on or near a non-linear manifold in the data space. For example, modeling the set of points that lie very close to the surface of a sphere only requires a few parameters using an appropriate model class, but it requires a very large number of diagonal Gaussians or a fairly large number of full-covariance Gaussians. Therefore, other types of model may work better than GMMs for acoustic modeling if they can more effectively exploit information embedded in a large window of frames.

Artificial neural networks trained by back-propagating error derivatives have the potential to learn much better models of data that lie on or near a non-linear manifold. In fact two decades ago, researchers achieved some success using artificial neural networks with a single layer of non-linear hidden units to predict HMM states from windows of acoustic coefficients. At that time, however, neither the hardware nor the learning algorithms were adequate for training neural networks with many hidden layers on large amounts of data and the performance benefits of using neural networks with a single hidden layer were not sufficiently large to seriously challenge GMMs. As a result, the main practical contribution of neural networks at that time was to provide extra features in tandem([21], Hermansky et al., 2000: 1635-1638.) or bottleneck systems.

Over the last few years, advances in both machine learning algorithms([22], Hinton and Osindero, 2006: 1527-1554.) and computer hardware have led to more efficient methods for training deep neural networks (DNNs) that contain many layers of non-linear hidden units and a very large output layer. The large output layer is required to accommodate the large number of HMM states that arise when each phone is modelled by a number of different ”triphone” HMMs that take into account the phones on either side. Even when many of the states of these triphone HMMs are tied together,

there can be thousands of tied states. Using the new learning methods, several different research groups have shown that DNNs can outperform GMMs at acoustic modeling for speech recognition on a variety of datasets including large datasets with large vocabularies([23], Hinton et al., 2012: 82-97.).

In this chapter, we will first introduce the basics of the DNN model, then describe how to use DNN to substitute the GMM, forming the DNN-HMM framework. Then we will review the DNN training(BP) and pre-training(RBM) algorithm.

3.1 Introduction to the Deep Neural Network

Neural networks, NNs(or artificial neural networks, ANNs) are designed to mimic the human brain, in which a lot of neurons are connected together to form a network. A lot of theoretical works have proved the modeling capability of ANN([24], Cybenko, 1989: 303-314.)([25], Hornik et al., 1989: 359-366.). There are very many classes of ANNs, feed-forward neural networks, recurrent neural networks(RNNs), convolutional neural networks(CNNs)([26], Abdel-Hamid et al., 2013: 3366-3370.), etc. In this work, we use the feed-forward neural network, the class of ANNs that is most widely used in automatic speech recognition. In a feed-forward neural network, the input feature vector under-go a series of linear or non-linear transform when propagating in the network, until it reaches the final output layer. Deep neural network(DNN) is a kind of feed-forward neural network with two or more hidden layers. "Deep" here means the NN has many layers, a deep structure.

First we introduce the basic building block of a neural network, a neuron. A neuron is connected to other neurons(e.g. the neurons in the previous layer), and take their outputs as its own inputs. Each connection has a certain **weight** to model the strength of the connection. The computation of a neuron is illustrated in figure 3.2. The computation in a neuron is very simple, the input signals are first added up and fed into the activation function. In this thesis, the activation is chosen to be the *sigmoid* function,

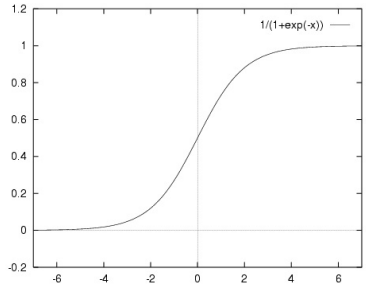

Figure 3.1 The sigmoid function

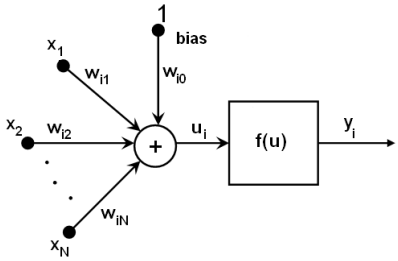

Figure 3.2 A neuron

which is mostly widely used:

$$f(x) = \frac{1}{1+e^{-x}}$$
$$f'(x) = f(x)(1-f(x)) \tag{3-1}$$

So, if we denote inputs as x_i and output as y, the computation of a neuron can be formalized as:

$$y = sigmoid(\sum_i x_i + b) \tag{3-2}$$

where b is a bias term.

The *sigmoid* function is plotted in figure 3.1. It is very much like a step function, the same like a human neuron. The main benefit of the sigmoid function is that it is derivable. We need to mention that other type of neurons can be used to induce non-linearity. For example, rectified linear unit has achieved huge success in ASR community([27], Dahl et al., 2013: 8609-8613.).

A deep neural network can be divided into many layers and each neuron belongs to a layer, as depicted in figure 3.3. The first is the input layer and the last is the output layer, we call the layers in the middle "hidden layers"(because users only care about the

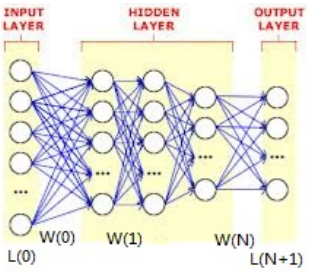

Figure 3.3 Layers of a deep neural network

input and output layers). Each neuron is connected to all the neurons in the previous layer and we call it "fully-connected". For a fully connected DNN, we can arrange the connection weights between two layers into a matrix **W**, we call it transformation matrix.

In the DNN-HMM framework, we use DNN as a classification model, which means each neuron on the output layer represents a label(states in the HMM model) and the DNN outputs the posterior probability of each label given the input feature vector. To be specific, each frame of the speech utterance is fed as a input vector **o** into the DNN, through each layer, the vector goes through a series of linear and non-linear transformation, finally in the output layer we get the posterior probability density $P(s|\mathbf{o})$. And because we need to ensure that the output layer is a probability density, a soft-max operation is forced on the output layer. The soft-max operation takes a input vector **x**, and transform it into a discrete probability distribution P, which is defined as follows:

$$P(i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{3-3}$$

Now we can formalize the propagation of a DNN, following figure 3.3, denote the input feature vector **o** also as $\mathbf{y}^{(0)}$ and assume there are L hidden layers, then activations on each layer can be calculated by activations on its previous layer:

$$y_i^l = sigmoid(\sum_j w_{ij}^{l-1} y_j^{l-1} + b_i^l) \tag{3-4}$$

or it can all be written in matrix form:

$$\mathbf{y}^l = sigmoid(\mathbf{W}^{l-1}\mathbf{y}^{l-1} + \mathbf{b}) \tag{3-5}$$

but here $sigmoid$ means element-wise sigmoid function. And finally, a soft-max operation is imposed on $\mathbf{y}^L$ to get the probability distribution $P(s|\mathbf{o})$ as output.

Here we define an important notation that will be used further in this thesis, which is C, the complexity of a DNN model. It is the number of all the parameters in the transformation matrices, if we denote number of neurons on each layer as L_i, then C can be formalized as:

$$C = \sum_{i=0}^{i=N} L_i L_{i+1} \tag{3-6}$$

It can be easily seen from (3-4) that time complexity of propagating a feature vector through a DNN can be approximated by C. So, for decoding a data-set with feature set O in ASR, the time spent on DNN computation can be approximated by:

$$T_{DNN}(O) = |O| \cdot O(C) \tag{3-7}$$

3.2 The DNN-HMM framework

The DNN-HMM ASR framework is very much like the GMM-HMM framework, except that the GMM models for each state in the HMM are thrown away, and they are modeled by a single DNN model, as depicted in figure 3.4.

The input to the DNN is feature vectors, and the neurons on the output layer represent HMM states(tied states). It is common practise to concatenate the previous and succeeding five along with the current frame to form an input vector. For example, if the feature is 39-dimension PLP, the input layer size will be $39 \times 11 = 429$.

As noted, GMM is a generative model which gives $p(\mathbf{o}|s)$, but DNN is used as a discriminative model which outputs $P(s|\mathbf{o})$. To embed the DNN into the framework,

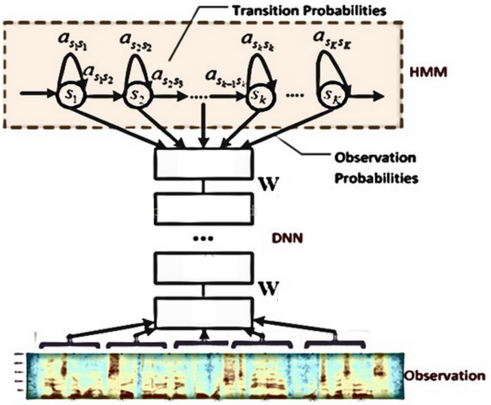

Figure 3.4 The DNN-HMM framework

we need to use the *Bayes Formula*:

$$p(\mathbf{o}|s) = \frac{P(s|\mathbf{o})p(\mathbf{o})}{P(s)} \tag{3-8}$$

where $p(\mathbf{o})$ is a constant and the prior probability $P(s)$ is usually computed from the statistics from the training data reference.

In the next sections, we will focus on how to train a DNN model.

3.3 RBM-pretraining

3.3.1 The Restricted Boltzmann Machine

Directly training a randomly-initialized Deep Neural Network with stochastic is difficult because the training could easily fall into a local-optima. So, not until recently researchers only use shallow neural networks(with very few hidden layers), otherwise we can only get DNN with very poor performance. Recently, with the introduction of DBN-DNN([22], Hinton and Osindero, 2006: 1527-1554.), in which the DNN is first trained in an unsupervised manner as a stack of Restricted Boltzmann Machines, the DNN training got a huge performance improvement and out-performed the classical GMM-HMM system in ASR systems. In this section, we will go through the RBM-pretraining in detail.

A Restricted Boltzmann Machine(RBM) consists of one hidden layer and one vis-

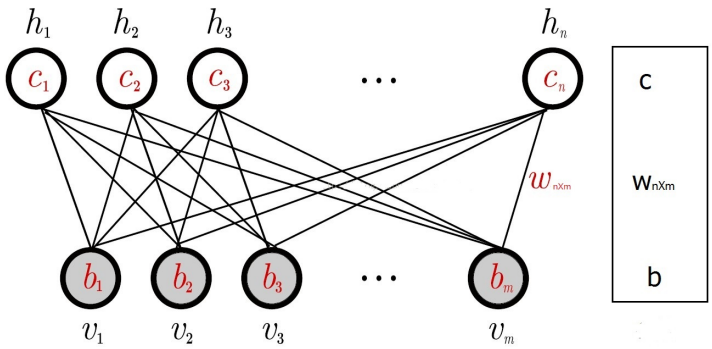

Figure 3.5 Restricted Boltzmann Machine

ible layer, showed in figure 3.5. It has a visible layer bias vector **b**, a hidden layer bias vector **c** and a weight matrix between the two layers **W**, so the parameter set is $\theta = (\mathbf{b}, \mathbf{c}, \mathbf{W})$. We call it "Restricted" because there are no connections within each layer.

Note that for ease of illustration, in this section we assume **h** and **v** are all **discrete binary variables**.

Basically, The RBM can be viewed as a generative probabilistic model, given **h** and **v**, the *energy* is defined by

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h} \tag{3-9}$$

Then we define $P(\mathbf{v}, \mathbf{h})$ to be

$$P(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \tag{3-10}$$

Here we pre-define $\gamma_i(\mathbf{v}, h_i) = -(c_i + \mathbf{v}^T W_{*i})h_i$(part related to h_i in the energy), which would be useful in proofs below.

Now, from (3-10) we can easily infer $P(h_i = 1|\mathbf{v})$:

$$\begin{aligned} P(h_i = 1|\mathbf{v}) &= \frac{P(h_i = 1, \mathbf{v})}{P(\mathbf{v})} \\ &= \frac{\sum_{\mathbf{h}\in(\ldots,h_i=1,\ldots)} e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\tilde{\mathbf{h}}} e^{-E(\mathbf{v},\tilde{\mathbf{h}})}} \\ &= \frac{e^{-\gamma_i(\mathbf{v},h_i=1)} \sum_{\mathbf{h}_{-i}} e^{-E(\mathbf{v},\mathbf{h}_{-i})}}{\sum_{\tilde{h_i}} e^{-\gamma_i(\mathbf{v},\tilde{h_i})} \sum_{\tilde{\mathbf{h}}_{-i}} e^{-E(\mathbf{v},\tilde{\mathbf{h}}_{-i})}} \\ &= \frac{e^{-\gamma_i(\mathbf{v},h_i=1)}}{\sum_{\tilde{h_i}} e^{-\gamma_i(\mathbf{v},\tilde{h_i})}} \end{aligned} \tag{3-11}$$

where $E(\mathbf{v}, \mathbf{h}_{-i}) = E(\mathbf{v}, \mathbf{h}) - \gamma_i(\mathbf{v}, h_i)$.Also, since h_i is assumed to be a binary variable, from (3-11) we can easily get

$$P(h_i = 1|\mathbf{v}) = sigmoid(c_i + \sum_j w_{ji} v_j) \tag{3-12}$$

where the *sigmoid* function is

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{3-13}$$

RBM has a nice property called *independence*, which means that given $\mathbf{v}$, the output probability of each hidden unit are conditionally independent and vice versa. Using (3-

11), it can be proven below:

$$\begin{aligned} P(\mathbf{h}|\mathbf{v}) &= \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\tilde{\mathbf{h}}} e^{-E(\mathbf{v},\tilde{\mathbf{h}})}} \\ &= \frac{e^{\mathbf{b}^T\mathbf{v}+\mathbf{c}^T\mathbf{h}+\mathbf{v}^T\mathbf{W}\mathbf{h}}}{\sum_{\tilde{\mathbf{h}}} e^{\mathbf{b}^T\mathbf{v}+\mathbf{c}^T\tilde{\mathbf{h}}+\mathbf{v}^T\mathbf{W}\tilde{\mathbf{h}}}} \\ &= \frac{e^{\mathbf{c}^T\mathbf{h}+\mathbf{v}^T\mathbf{W}\mathbf{h}}}{\sum_{\tilde{\mathbf{h}}} e^{\mathbf{c}^T\tilde{\mathbf{h}}+\mathbf{v}^T\mathbf{W}\tilde{\mathbf{h}}}} \\ &= \frac{\prod_i e^{-\gamma_i(\mathbf{v},h_i)}}{\prod_i \sum_{\tilde{h_i}} e^{-\gamma_i(\mathbf{v},\tilde{h_i})}} \\ &= \prod_i \frac{e^{-\gamma_i(\mathbf{v},h_i)}}{\sum_{\tilde{h_i}} e^{-\gamma_i(\mathbf{v},\tilde{h_i})}} \\ &= \prod_i P(h_i|\mathbf{v}) \end{aligned} \tag{3-14}$$

Now we move on to the unsupervised training of RBM. We have already mentioned that RBM can be viewed as a generative model, so given a set of seen data set V, we apply the maximum likelihood estimation (MLE), so the loss function is log-likelihood, here for simplicity, we only consider one single $\mathbf{v}$.

$$L_{\mathbf{v}}(\theta) = log(P(\mathbf{v})) = log \sum_{\mathbf{h}} P(\mathbf{v},\mathbf{h}) = -F(\mathbf{v}) - log(\sum_{\mathbf{v}} e^{-F(\mathbf{v})}) \tag{3-15}$$

where we define a free energy term $F(v) = -log(\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})})$.

Like the fine-tuning of DNN, RBM-pretraining is also a stochastic gradient descent procedure, in which the parameters are updated every mini-batch. Now, we calculate

gradient of $L_{\mathbf{v}}$ with respect to w_{ij}.

$$\begin{aligned}
\frac{\partial L_{\mathbf{v}}}{\partial w_{ij}} &= \frac{-\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} v_i h_j}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} + \frac{\sum_{\mathbf{v}} e^{-F(\mathbf{v})} (\frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} v_i h_j}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}})}{\sum_{\mathbf{v}} e^{-F(\mathbf{v})}} \\
&= -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}) v_i h_j + \frac{\sum_{\mathbf{v}} e^{-F(\mathbf{v})} (\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}) v_i h_j)}{\sum_{\mathbf{v}} e^{-F(\mathbf{v})}} \\
&= -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}) v_i h_j + \frac{\sum_{\mathbf{v}} P(\mathbf{v}) (\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}) v_i h_j)}{\sum_{\mathbf{v}} P(\mathbf{v})} \\
&= -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}) v_i h_j + \sum_{\mathbf{v},\mathbf{h}} P(\mathbf{h},\mathbf{v}) v_i h_j \\
&= - < v_i h_j >_{data} + < v_i h_j >_{model}
\end{aligned} \tag{3-16}$$

Using the *independence* property, the first term $< v_i h_j >_{data}$ can be efficiently calculated:

$$\begin{aligned}
\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}) v_i h_j &= \sum_{\mathbf{h}} \prod_k P(h_k|\mathbf{v}) v_i h_j \\
&= v_j (P(h_i = 0|\mathbf{v}) \cdot 0 + P(h_i = 1|\mathbf{v}) \cdot 1) \prod_{k \neq i} (P(h_k = 0|\mathbf{v}) + P(h_k = 1|\mathbf{v})) \\
&= P(h_i = 1|\mathbf{v}) v_j
\end{aligned} \tag{3-17}$$

which can be efficiently calculated by (3-12). However, using the same technique we can only reduce $< v_i h_j >_{model}$ to $\sum_{\mathbf{v}} P(\mathbf{v}) P(h_i = 1|\mathbf{v}) v_j$, which still needs exponential computations.

This problem is attacked by contrastive divergence([28], Hinton, 2002: 1771-1800.)([29], Fischer and Igel, 2012: 14-36.), which will be discussed in the next section.

3.3.2 Contrastive divergence

First we need to introduce a class of generative model called *product of experts*, if it can be dissembled in to m factors:

$$P(\mathbf{v}|\theta) = \frac{\prod_m P_m(\mathbf{v}|\theta_m)}{\sum_{\tilde{\mathbf{v}}} \prod_m P_m(\tilde{\mathbf{v}}|\theta_m)} \tag{3-18}$$

And RBM can be viewed as a *product of experts* because

$$P(\mathbf{v}) = \frac{1}{Z}\prod_j e^{b_j v_j}\prod_i (1 + e^{c_i + \sum_j w_{ij} v_j}) \tag{3-19}$$

where Z is the normalizing factor.And we have

$$\frac{\partial log P(\mathbf{v}|\theta)}{\theta_m} = \frac{\partial log P_m(\mathbf{v}|\theta_m)}{\theta_m} - \sum_{\tilde{\mathbf{v}}} P(\tilde{\mathbf{v}}|\theta)\frac{\partial log P_m(\tilde{\mathbf{v}}|\theta_m)}{\theta_m} \tag{3-20}$$

Given a data distribution Q^0 and a generative model P, maximizing the log likelihood of the data(over the data distribution) is equivalent to minimizing the Kullback-Liebler divergence between the data distribution Q^0 and the equilibrium distribution Q^∞ which is produced by prolonged Gibbs sampling from the generative model.

$$Q^0||Q^\infty = \sum_{\mathbf{v}} Q^0(\mathbf{v}) log Q^0(\mathbf{v}) - \sum_{\mathbf{v}} Q^0(\mathbf{v}) log Q^\infty(\mathbf{v}) = H(Q^0) - < log Q^\infty(\mathbf{v}) >_{Q^0} \tag{3-21}$$

Where $||$ denotes Kullback-Liebler divergence. In (3-21) the first term is the entropy of the data distribution which is a constant, and the second term is the log-likelihood. Now we take gradient of the second term:

$$\frac{\partial Q^0||Q^\infty}{\partial \theta_m} = < \frac{\partial log P_m(\mathbf{v}|\theta_m)}{\partial \theta_m} >_{Q^0} - < \frac{\partial log P_m(\tilde{\mathbf{v}}|\theta_m)}{\partial \theta_m} >_{Q^\infty} \tag{3-22}$$

From (3-17), we know that the first term is tractable, but the second term is not.

What contrastive divergence means is that, instead of minimizing $Q^0||Q^\infty$, we minimize the difference between $Q^0||Q^\infty$ and $Q^1||Q^\infty$, where Q^1 is the distribution reconstructed by one step of Gibbs sampling from the data distribution Q^0. So, we approximate $Q^0||Q^\infty$ by

$$Q^0||Q^\infty - Q^1||Q^\infty \tag{3-23}$$

Since Q^1 is closer to the final equilibrium distribution, it will always be positive.

The mathematical motivation of the contrastive divergence is that the intractable

part cancels out:

$$-\frac{\partial(Q^0||Q^\infty - Q^1||Q^\infty)}{\partial\theta_m} = < \frac{\partial logP_m(\mathbf{v}|\theta_m)}{\partial\theta_m} >_{Q^0} - < \frac{\partial logP_m(\tilde{\mathbf{v}}|\theta_m)}{\partial\theta_m} >_{Q^1} + \frac{\partial Q^1}{\partial\theta_m}\frac{\partial Q^1||Q^\infty}{\partial Q^1} \tag{3-24}$$

The third term in (3-24) is hard to compute, but extensive simulations showed it can be ignored.

This is called the *CD-1* approximation, in which Q^1 is used, more Gibbs sampling steps can be conducted to get Q^k, called *CD-k*. In experiments we found that *CD-1* is enough to get good results. The complete process of RBM-training is formalized in algorithm 1.

Algorithm 1 k-step contrastive divergence

input: $RBM(\mathbf{b}, \mathbf{c}, \mathbf{W})$ and training batch V.
output: gradient approximation Δw_{ij}, Δb_j, Δc_i, for $i = 1, ..., n$ and $j = 1, ..., m$.

init $\Delta w_{ij} = \Delta b_j = \Delta c_i = 0$
for $v \in V$ **do**
  $\mathbf{v}^{(0)} \leftarrow \mathbf{v}$
  for $t = 0, ..., k-1$ **do**
    for $i = 1, ..., n$ **do**
      sample $h_i^{(t)} \sim P(h_i|\mathbf{v}^{(t)})$
    end for
    for $j = 1, ..., m$ **do**
      sample $v_j^{(t+1)} \sim P(v_j|\mathbf{h}^{(t)})$
    end for
  end for
  for $i = 1, ..., n, j = 1, ..., m$ **do**
    $\Delta w_{ij} \leftarrow \Delta w_{ij} + P(h_i = 1|\mathbf{v}^{(0)}) \cdot v_j^{(0)} - P(h_i = 1|\mathbf{v}^{(k)}) \cdot v_j^{(k)}$
    $\Delta b_j \leftarrow \Delta b_j + v_j^{(0)} - v_j^{(k)}$
    $\Delta c_i \leftarrow \Delta c_i + P(h_i = 1|\mathbf{v}^{(0)}) - P(h_i = 1|\mathbf{v}^{(k)})$
  end for
end for

3.3.3 Stacked RBM

For DNN pretraining, we view the DNN as stacked Restricted Boltzmann Machines. The training process go through a greedy layerwise manner, as illustrated in figure 3.6.

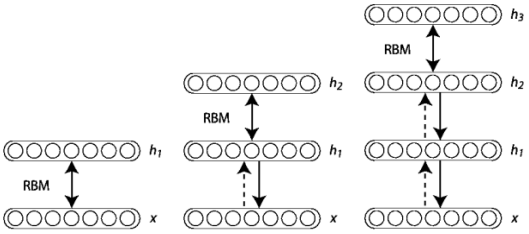

Figure 3.6 greedy layerwise training of RBM

First we directly use all the training data to train a single-layer RBM, using algorithm 1. Then, for every hidden layer in the DNN, another RBM is stacked on the existing model. In further training, we fix the existing model and train only the new top RBM(we use the output of the existing model as the input). The RBM is stacked until the final hidden layer is reached. In the last step the output layer is added upon the RBM-pretrained DNN, the parameters between the last two layers are randomly generated.

3.4 The back-propagation algorithm

In this section we move onto the training of a DNN model. Assume we have a training data set of n frames, and we've got force-aligned results t_i which indicates which HMM state each frame belongs to. For a classification, the most commonly used objective function is the negative log-likelihood. Thanks to the final softmax operation on the output layer, the objective function can be written as:

$$L(\theta) = -\sum_i log(P(s = t_i|\mathbf{o}_i)) \tag{3-25}$$

This objective function is also named *cross-entropy*, because we can construct a target distribution P' for each frame, where only $P'(s = t_i)$ is equal to one and otherwise zero. Then the objective function can be viewed as a summation of cross-entropy between two distributions of each frame.

Note that this is a full-batch training objective function, in which all training frames are taken into consideration. In experiments, it is found that the convergence speed of full-batch training is very slow. Instead, people use stochastic gradient descent(SDS).

SDS means that in each epoch a small amount of data is randomly selected from the whole dataset and used to update the model. The choice of the size of the mini-batch is however, a trade-off between convergence speed and computational efficiency(because nowadays DNN training is done on GPU, which has massive parallelism).

The training algorithm of DNN is named back-propagation(BP) because the error is propagated from the output layer down to the input layer. And thanks to the layer-wise structure of DNN, each parameter's gradient is only related the error on its next layer. We elaborate the back-propagation process below.

Here for simplicity, we only take the derivative with respect to one single frame which has feature $\mathbf{o}$ and is assigned label t, since we only need to add gradient for each frame up to get the gradient of the mini-batch. Following notations in section 3.1, we assume $\mathbf{y}^0$ is the input feature vector, also the activations on the input layer of DNN. Before we do back-propagation, a feed-forward pass of the input is needed. Now we first deduce the error on the output layer:

$$\begin{aligned}\frac{\partial L}{\partial y_i^{L+1}} &= \frac{-\partial log(\frac{e^{y_t^{L+1}}}{\sum_i e^{y_i^{L+1}}})}{\partial y_t^{L+1}} \\ &= \frac{e_i^{L+1}}{\sum_i e^{y_i^{L+1}}} - \frac{\partial y_t^{L+1}}{\partial y_i^{L+1}} \\ &= P(s=i|\mathbf{o}) - [t=i]\end{aligned} \tag{3-26}$$

where we define $[assertion]$ is equal to one when the assertion is true, and zero otherwise.

Now, use (3-4), we propagate the error to previous layers, for $1 \leq l \leq L$, first, the loss function can be written as a function of activations on layer $l+1$ because of the layer-wise structure of DNN:

$$\frac{\partial L}{\partial y_i^l} = \frac{\partial \hat{f}^l(\mathbf{y}^{l+1})}{\partial y_i^l} \tag{3-27}$$

Now we need to use the *chain rule* for multivariate composite function, which we formalize below:

Theorem 3.1. ***(chain rule for multivariate composite function)*** *We have a function*

$f(x_1, ..., x_n)$ and for each x_i we have $x_i = g_i(u_1, ..., u_m)$, and all functions are derivable, then

$$\frac{\partial f}{\partial u_i} = \sum_j \frac{\partial f}{\partial x_j} \cdot \frac{\partial g_j}{\partial u_i} \tag{3-28}$$

Using the chain rule, here we the propagate the error layer by layer:

$$\frac{\partial L}{\partial y_i^l} = \sum_j (\frac{\partial L}{\partial y_j^{l+1}} \cdot y_j^{l+1} \cdot (1 - y_j^{l+1}) \cdot w_{ji}^l) \tag{3-29}$$

Now we can deduce derivative of every parameters, again using the *chain rule*:

$$\begin{aligned} \frac{\partial L}{\partial w_{ij}^l} &= \frac{\partial L}{\partial y_i^{l+1}} \cdot \frac{\partial y_i^{l+1}}{\partial w_{ij}^l} \\ &= \frac{\partial L}{\partial y_i^{l+1}} \cdot y_i^{l+1} \cdot (1 - y_i^{l+1}) \cdot y_j^l \\ \frac{\partial L}{\partial b_i^l} &= \frac{\partial L}{\partial y_i^{l+1}} \cdot \frac{\partial y_i^{l+1}}{\partial b_i^l} \\ &= \frac{\partial L}{\partial y_i^{l+1}} \cdot y_i^{l+1} \cdot (1 - y_i^{l+1}) \end{aligned} \tag{3-30}$$

From the formulas of BP algorithm above, it can be easily deduced that computational complexity of doing BP can also be approximated by $O(C)$, where C is the complexity of the model, defined in (3-6). It's hard to approximate the time of training DNN on GPU according to the model complexity because of vast parallelism, but still, smaller model complexity means faster training speed.

3.4.1 Training strategy

As discussed, in actual DNN training, we randomly take a small amount of data, use the BP algorithm to calculate the gradients of parameters and update the model. We need, however, to set a learning rate α for each epoch. It is also a best practise to induce a L2-regularization term in the objective function to prevent parameters from being to large, let the regularization term's coefficient be β. Then the updating formula can be written as follows:

$$\theta_{new} = \theta + \alpha \cdot \frac{\partial L}{\partial \theta} - \beta \cdot \theta \tag{3-31}$$

For gradient, the momentum method is used for accelerating training convergence. Set a momentum term m. Then for every mini-batch, m times the old gradient will be added to the gradient of the current mini-batch.

α needs to be tuned through the whole training process according to a certain training strategy. In the experiments of this thesis, a learning rate annealing and early stopping strategy([1], rahman Mohamed et al., 2012: 14-22.) is used. One tenth of the training data is randomly chosen to form a held-out cross validation(CV) data-set, it won't be used in the training, and after each iteration the model is tested on the CV set for frame accuracy. We start with a large learning rate, and begins to decay it when the CV accuracy improvement becomes small. The training stops after the CV accuracy improvement becomes smaller than a certain threshold.

3.5 Link with the GMM-HMM system

Even if now the DNN-HMM framework has shown better performance, the training is not GMM-free, which means it still needs a GMM-HMM baseline system. The GMM-HMM system will be used to give state-level alignment for training data to be used for DNN training, also the DNN will use the same tied-state as GMM. As for decoding, the HMM model remains the same, while the GMM is replaced by the DNN model. Another interesting difference is that in the old system each HMM state has a GMM, but now a single DNN models all the states.

3.6 Summary

In this chapter we go over the basics of the DNN model and the DNN-HMM ASR framework, in which the *Bayes formula* is used to convert the posterior probability from the DNN model. The DNN model can not be randomly initialized for supervised training because experiments showed that the training is easy to fall into a local optima. We introduced the RBM-pretraining algorithm in this chapter, in which the DNN is first viewed as a stack of Restricted Boltzmann manchines and trained in a unsupervised manner. All experiments in this thesis uses RBM-pretraining. We need to mention here

are other techniques found efficient to initialize to DNN. One is called discriminative pre-training, or layerwise BP([3], Bengio et al., 2007: 153-160.), in which the DNN also grows layer by layer, but each time a new output layer is added and the growing model is fine-tuned for some iterations. Then we reviewed the classic back-propagation algorithm for DNN training and the training strategy we used in this work. Even with the help of GPU, DNN training on a fairly large data-set is intolerably slow, because larger data-set means we need to use a larger model. For example, a 300 hour English data set would need a 7-hidden-layer DNN with 2048 neurons on each hidden layer, and the training can take more than two weeks. Finally, we note that to train a DNN model a GMM-HMM baseline system is needed.

Chapter 4 Reshaping DNN by node-pruning

4.1 Exploiting the sparseness of DNN

As discussed in section 1.2, a typical DNN for LVCSR has huge amount of parameters. For example, to achieve state-of-art performance on the switchboard English task, we need a DNN with more than seven hidden layers with two thousand hidden neurons on each hidden layer. Therefore a typical DNN usually has more than 30 million(30M) parameters. The vast number of parameters make DNN space-consuming to store and time-consuming for computation. We have already analyzed the time complexity of DNN propagation in section 3.1. Therefore, after using DNN to achieve big performance improvement, researchers begin to think about how, without performance loss, to train a DNN using less time, or restructuring an existing baseline DNN to make it faster for decoding. This thesis focuses on restructuring DNN for fast decoding, in this section, we will review recent related works which exploit the sparseness of DNN.

4.1.1 Weight-truncation

As a pioneering work, Dong Yu tried to remove connections of DNN in order to reduce the number of parameters of DNN in ([9], Yu et al., 2012: 4409-4412.). This work is inspired by the weight magnitude of a typical DNN, as illustrated in figure 4.1. It can be observed that most of the weights' magnitude is below 0.1. In this work an algorithm is proposed to train a sparse DNN by removing the smaller weights. The algorithm is described in algorithm

Algorithm 2 Main steps to train a sparse DNN

input: Percentage q, amount of weight to reserve.

1: Train a fully connected DNN by sweeping through the full training set several times using labels from forced alignment.
2: Keep only the connections whose weight magnitudes are in top q.
3: Continue training the DNN with the sparseness pattern unchanged.

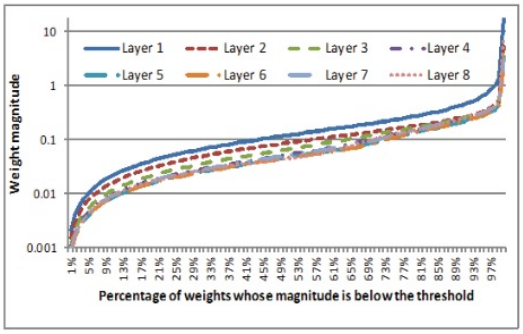

Figure 4.1 The weight magnitude distribution of a 7-hidden-layer DNN illustrated as the percentage of weights whose magnitude is below a threshold

The method is simple but effective, experiments show that on the SWB data-set, with a baseline DNN that has 7 hidden layers and 2048 hidden nodes on each layer, the number of parameters can be reduced to 29% without visible performance loss. This astonishing result proves that a sparseness constraint can be imposed in DNN training. We need to note that to indeed speed up DNN computation and storage, special implementation is required.

4.1.2 Singular Value Decomposition on weight matrices

A shortcoming of the weight-truncation is that the sparseness pattern is random so it requires special implementation for deployment. In section 3.1 we mentioned that the parameters in DNN can be arranged into transformation matrices and the computation of DNN can be viewed as a series of matrix-vector operation. ([10], Xue et al., 2013: 2365-2369.) proposed to use singular value decomposition on the transformation matrices.

Given a $m \times n$ matrix M, SVD decomposes M into:

$$M = U\Sigma V \tag{4-1}$$

where Σ is a diagonal matrix containing singular values. By setting a rank r SVD can decompose a $m \times n$ matrix into two small $m \times r$, $r \times n$ matrices by truncating the

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} u_{11} & \cdots & u_{1n} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mn} \end{bmatrix} \cdot \begin{bmatrix} \epsilon_{11} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \epsilon_{kk} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & \epsilon_{nn} \end{bmatrix} \cdot \begin{bmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{n1} & \cdots & v_{nn} \end{bmatrix}$$

$$\approx \begin{bmatrix} u_{11} & \cdots & u_{1n} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mn} \end{bmatrix} \cdot \begin{bmatrix} \epsilon_{11} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \epsilon_{kk} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 \end{bmatrix} \cdot \begin{bmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{n1} & \cdots & v_{nn} \end{bmatrix}$$

$$= \begin{bmatrix} u_{11} & \cdots & u_{1k} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mk} \end{bmatrix} \cdot \begin{bmatrix} \epsilon_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \epsilon_{kk} \end{bmatrix} \cdot \begin{bmatrix} v_{11} & \cdots & v_{1k} \\ \vdots & \ddots & \vdots \\ v_{k1} & \cdots & v_{kk} \end{bmatrix}$$

Figure 4.2 SVD rank truncation

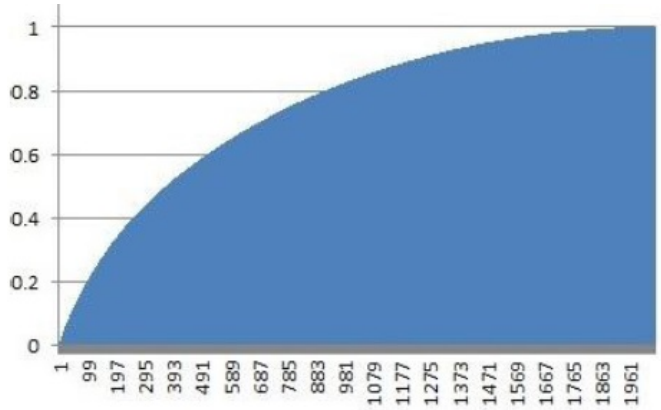

Figure 4.3 Distribution of singular values of a 2048×2048 transformation matrix in a 5-hidden-layer DNN

middle singular value matrix, as illustrated in figure 4.2, and formalized as:

$$\begin{aligned} M_{m\times n} &= U_{m\times n}\Sigma_{n\times n}V_{n\times n} \\ &\approx U_{m\times r}\Sigma_{r\times r}V_{r\times n} \\ &= U_{m\times r}V'_{r\times n} \end{aligned} \tag{4-2}$$

SVD has been proved to provide a good low-rank approximation of a matrix:

Theorem 4.1. *Given a matrix M, SVD gives matrix $\tilde{M}$ that minimizes the Frobenius norm of the difference between M and $\tilde{M}$ under the constraint that $rank(\tilde{M}) = r$.*

A plot from ([10], Xue et al., 2013: 2365-2369.) is shown in figure 4.3, it can be observed that most singular values are relatively small comparing to the largest ones.

The SVD restructuring algorithm is very much like algorithm 2, take a fully trained DNN, we do SVD restructuring of rank r, the restructured DNN would lose some ac-

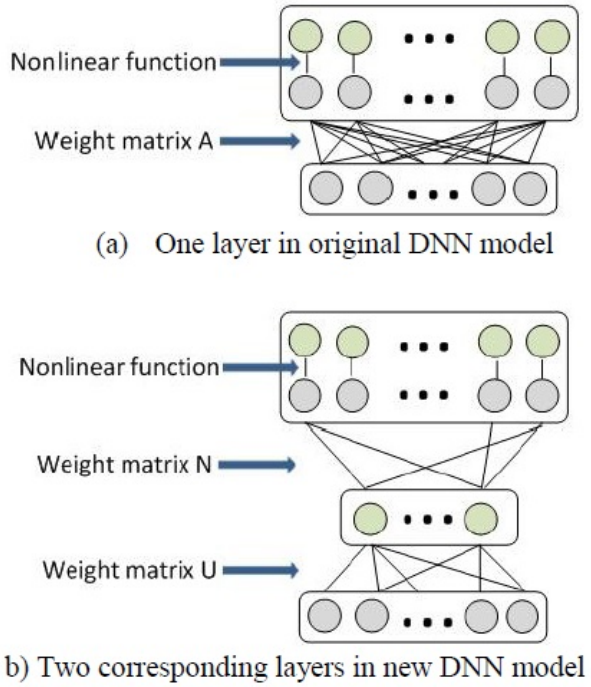

Figure 4.4 SVD restructuring of DNN

curacy, so we train it a few more iterations. In ([10], Xue et al., 2013: 2365-2369.) experiments are conducted on a Microsoft internal task, using a DNN of 5 hidden layers of 2048 neurons on each hidden layer. Experiments show that rank-192 SVD can be used to restructure all transformation matrices, reducing the model complexity to 25%.

4.2 Reshaping DNN by node-pruning

In the last section we reviewed recent works on restructuring DNN for fast decoding, we mentioned that they all focus on weight-level restructuring, so the resulting deep neural network is not ”fully-connected” anymore. Node-pruning, however, removes hidden neurons, along with its connections and bias, out of the original DNN, so the resulting DNN is still a fully-connected DNN, except that the number of neurons on each hidden layer is different from the original DNN, which is why node-pruning can be called ”reshaping” DNN. This brings two big benefits: the resulting DNN can be directly used for decoding or further training without implementation modification; node-pruning can be seamlessly combined with other restructuring techniques to achieve fur-

ther model complexity reduction.

The basic assumption of the proposed node-pruning framework is that the DNN prior to pruning has redundant nodes. Since there is no sound theoretical guidance of choosing the the shape(number of hidden nodes of each layer) or scale to achieve optimal DNN structure, people normally employ certain level of redundancy to guarantee good decoding performance. Sometimes, the redundancy can be large, for example, DNN of 2048 nodes per layer may be adopted for the TIMIT(a small data set) experiments. This practical choice makes the assumption in this paper reasonable. Therefore, a pre-requisite of *node-pruning* is a sufficiently wide and deep neural network, referred to as *original DNN*. In this thesis, the original DNN is the one after parameter fine-tuning using the back-propagation (BP) algorithm [1].

Node-pruning is only applied to hidden nodes in this work. Prior to pruning, it is necessary to evaluate the importance of each hidden node. In this paper, three *node importance functions* are proposed as below.

- *Entropy*

  The *entropy* importance function directly examines the activation distribution of each node. A node is *activated* if the output value is greater than a threshold (0.5 is used in this paper). Let $|\mathbf{O}|$ be the total number of input frame set $\mathbf{O}$, which is chosen to be the whole training dataset in this work. Let $a_i^l(\mathbf{O})$ and $d_i^l(\mathbf{O})$ be the total number of frames which activate or de-activate node i of layer l respectively ($a_i^l(\mathbf{O}) + d_i^l(\mathbf{O}) = |\mathbf{O}|$), the *entropy* importance function of node (l, i) is then defined as

$$\mathcal{J}_e(l,i) = \frac{d_i^l(\mathbf{O})}{|\mathbf{O}|}\log_2(\frac{d_i^l(\mathbf{O})}{|\mathbf{O}|}) + \frac{a_i^l(\mathbf{O})}{|\mathbf{O}|}\log_2(\frac{a_i^l(\mathbf{O})}{|\mathbf{O}|})$$

  The intuition is that if one node's outputs are almost identical on all training data, these outputs do not generate variations to later layers and consequently the node may not be useful. The *entropy* importance function is similar to the ones

[1] Some early experiments on TIMIT showed that using the RBM-pretrained DNN as the original DNN resulted in noticeable performance degradation. Hence it is not further investigated in the paper.

proposed in early works([15], Yamamoto et al., 1993: ?.), ([30], Sietsma and Dow, 1991: 67-69.) since it focuses on the outputs of a node.

- *Output-weights Norm (onorm)*

  onorm measures the importance of a node (l, i) by the average L1-norm of the weights of its outgoing links, which is formulated as

$$\mathcal{J}_o(l,i) = \frac{1}{N^{l+1}} \sum_{j=1}^{N^{l+1}} |w_{ij}^{l+1}|$$

  where N^{l+1} is the number of nodes of the layer $l+1$. The intuition is that a node's importance should be reflected by its related weights after the BP process.

- *Input-weights norm (inorm)*

  Similiar to *onorm*, *inorm* reflects the importance of a node by the average L1-norm of the weights of its incoming links, which is formulated as

$$\mathcal{J}_i(l,i) = \frac{1}{N^{l-1}} \sum_{j=1}^{N^{l-1}} |w_{ji}^{l}|$$

After training, a score is calculated for each hidden node using one of the above importance functions. Then all the nodes are sorted by their scores and nodes with less importance values are removed. Along with the removal of a node, all relevant incoming and outgoing links are also removed. It's worth noting that although every node removal will affect the scores of some other nodes, in this paper, all nodes are only examined once with the scores calculated using the original DNN.

Testing experiments on the TIMIT data-set in section 5.3.1 show that all three importance functions work well, and random node selection will give unstable performance. Since I don't have time and resource to try all three importance functions on LVCSR(swithboard) task, I choose the *onorm* importance function as the final choice.

In contrast to weight truncation or decomposition, direct pruning hidden nodes will result in significant performance degradation, as will be shown later in section 5.3.1.

Hence, the pruned DNN needs to be re-finetuned. A big starting learning rate is used in the re-finetuning, so the node-pruned DNN can be viewed as an initialization of DNN. This means if we apply node-pruning on a fully-trained DNN for fast decoding, node-pruning requires more training time. However, different from other methods, the reshaped DNN can be directly used without any code or resources modification during decoding. In section 5.4 we will apply node-pruning in the middle of DNN training in order to speed-up DNN training.

4.3 Combination with the SVD technique

As stated in section 4.2, node-pruning is complementary to weight matrices decomposition methods and may be combined to get better model complexity saving. Since the result of node-pruning is still a fully-connected normal DNN, we can still do SVD decomposition on the transformation matrices. Section 3.4.1 talks about the training strategy we use in this work, we use the same training strategy for re-training in node-pruning and in SVD decomposition. The complete restructuring steps is listed in algorithm 3.

Algorithm 3 Combination of node-pruning and SVD

input: n : how many nodes to prune, r : rank of SVD decomposition.

1: Train a fully connected DNN.
2: Sort all hidden neurons according to the *o-norm* importance function.
3: Remove n hidden neurons with least importance score from the original DNN.
4: Re-train the model with the node-pruned DNN as an initialization.
5: Apply rank-r SVD technique on all the transformation matrices(We apply rank-r SVD on a $m \times n$ matrix if and only if $m \times r + r \times n < m \times n$ since we want to reduce model complexity.)
6: Re-train the resulting DNN for another few iterations with a small learning rate.

4.4 Summary

In this chapter we first review some related work on restructuring DNN for reduce model complexity. These works have shown that sparseness constraint can be imposed on the DNN model without performance loss. Node-pruning is different from those

techniques because it reshapes DNN by removing hidden neurons. Three importance functions are proposed for sorting hidden neurons. Node-pruning has two big benefits: the resulting DNN can be directly used for decoding or further training without implementation modification; node-pruning can be seamlessly combined with other restructuring techniques to achieve further model complexity reduction.

Chapter 5 Experiments

In this chapter we will give experiment results about restructuring DNN. First we go over the HTK toolkit used to train the GMM-HMM model and the TNET toolkit used to do RBM-pretraining and DNN fine-tuning. The baseline GMM-HMM result is also reported in section 5.1. Then, node-pruning experiments are conducted on the TIMIT and switchboard English tasks, respectively and show its power of reducing redundancy of DNN model.

5.1 The Hidden Markov Model Toolkit

The core step in building GMM-HMM model is to initialize and train the HMM model and the corresponding GMMs on each state. Tools for feature extraction and decoding are also needed. The Hidden Markov Model Toolkit(HTK)([31], Young et al., 2006: ?.) contains a series of tools for building GMM-HMM ASR framework, and has already become the standard toolkit in the speech community. In this thesis, the GMM-HMM baseline models are trained using HTK.

The Hidden Markov Model Toolkit (HTK) is a portable toolkit for building and manipulating hidden Markov models. HTK is primarily used for speech recognition research although it has been used for numerous other applications including research into speech synthesis, character recognition and DNA sequencing. HTK is in use at hundreds of sites worldwide.

HTK consists of a set of library modules and tools available in C source form. The tools provide sophisticated facilities for speech analysis, HMM training, testing and results analysis. The software supports HMMs using both continuous density mixture Gaussians and discrete distributions and can be used to build complex HMM systems. The HTK release contains extensive documentation and examples.

The tools and their uses in HTK are listed in figure 5.1.

I report my GMM-HMM experiments on a 50 hour subset of switchboard English

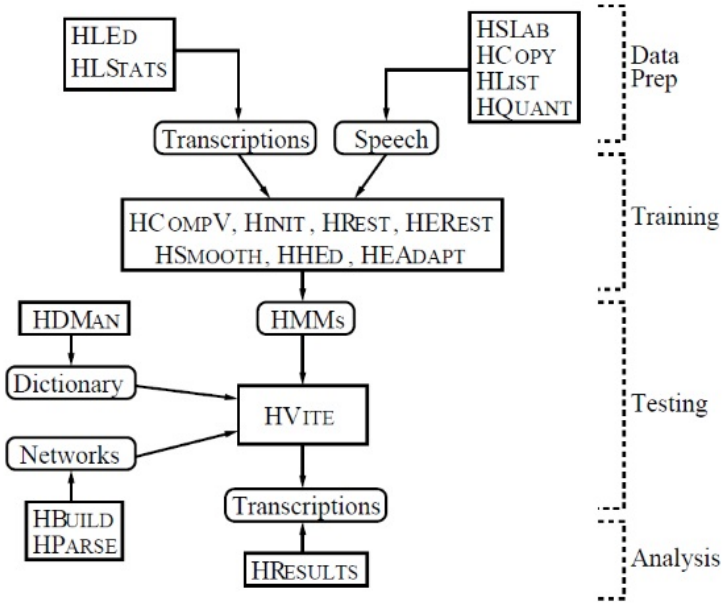

Figure 5.1 Tools and their uses in HTK

here, the feature is 39-dim PLP, the number of Guassian mixtures are 40, using three thousand tied states. The experiment setting is comparable with the DNN experiments in section . The word error rate(WER) on the switchboard (`swb`) part of the Hub5'00 data set is 37.3%. Since GMM-HMM training is not the focus of this thesis, I don't refer to details of the experiments here.

5.2 Neural Network Trainer TNET

DNN training is extremely time-consuming because of the huge model complexity. However, recent advances in hardware and software of GPU enables researchers to train DNN on GPU, utilizing the massive parallelism power of GPU.

TNET(`http://speech.fit.vutbr.cz/software/neural-network-trainer-tnet`) is a fast tool for parallel training of neural networks for classification written in C++. There are two sets of tools, the CPU and GPU tools. The fast CPU training is based on multithread data-parallelization approach. The GPU training is implemented by means of CUDA GPGPU. The training algorithm is mini-batch Stochastic Gradient Descent.

In my experiments, I use TNET to do the RBM-pretraining and fine-tuning of DNN on GPU. The TNET is compatible with HTK(can read master label file, feature files generated from HTK).

Here I give the specific parameters about the DNN training strategy. The mini-batch size is 128 frames. We use $\alpha = 0.8$ as the starting learning rate. On every iteration frame accuracy(in percentage) of the trained model is tested on the CV set, once the CV accuracy improvement is less than 0.5%. We start to decay the learning rate by factor 0.6 on every iteration. For gradient, the momentum factor m is set to 0.9. The training stops once the CV improvement is less than 0.01.

5.3 Node-pruning for fast decoding

Various aspects of the node-pruning approach was first investigated in detail using the TIMIT corpus. We choose the *o-norm* importance function from experiments on TIMIT, the approach was applied to a LVCSR task: switchboard English. Combination with SVD based weight decomposition was then performed and achieved the largest reduction of DNN complexity without significant performance degradation.

Note that *"complexity"* of DNN here refers to the total number of parameters of the weight transition matrices, as defined in (3-6).which was also adopted in ([32], Vinyals and Morgan, 2013: 114-118.). The *complexity* is proportional to the computational cost of calculating state posteriors from DNN and directly reflects decoding speed of a DNN-based speech recognition system.

5.3.1 Experiments on TIMIT

The proposed node-pruning approach was first investigated on a phone recognition task using the TIMIT corpus. The standard training set consisting of 462 speakers was used for RBM pre-training and DNN fine-tuning, and the 24-speaker core test set was used for evaluation. 40-dimensional Mel-scale filter bank coefficients along with their first and second derivatives were used as features. 11 consecutive frames were concatenated to form the input vectors of DNN. 183 target classes labels (3 states/phone,

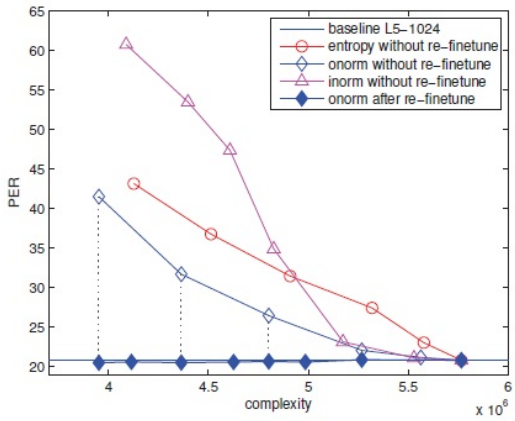

Figure 5.2 Performance of node-pruned DNN without re-finetuning on TMIT

61 phones) and mini-batches of size 128 were used during training.

The RBM-pretrained DNN was fine-tuned with cross-entropy objective function, along with a L2-norm weight-dacay term of coefficient 10^{-6}. After decoding, the 61 phone classes were mapped to a standard set of 39 classes for scoring. A 5-layer DNN with 1024 nodes per layer was trained as baseline. The phone error rate (PER) is 20.79% and the complexity is 5.76M parameters.

To investigate how much damage would node-pruning do to the baseline DNN, results of node-pruning with and without re-finetuning were compared.

100, 250, 500, 750 hidden nodes were pruned using the three importance functions defined in section 4.2 respectively. Figure 5.2 shows that without retraining, PER of the pruned DNN (1000 nodes pruned) increases from 20.79% to around 40%. The severe performance degradation has not been observed in weight decomposition methods([10], Xue et al., 2013: 2365-2369.). This implies that node pruning changes DNN structure more significantly and retraining is necessary to obtain comparable performance to the original DNN. This is justified by the performance of the *o-norm* node pruning with retraining in figure 5.2, which yielded almost the same PER as the baseline DNN.

To investigate the impact of using different importance functions, the three importance functions defined in section 4.2 and a random score function were evaluated respectively. The results are illustrated in figure 5.3. It is shown that node-pruning with the proposed importance functions can get the same or even better performance com-

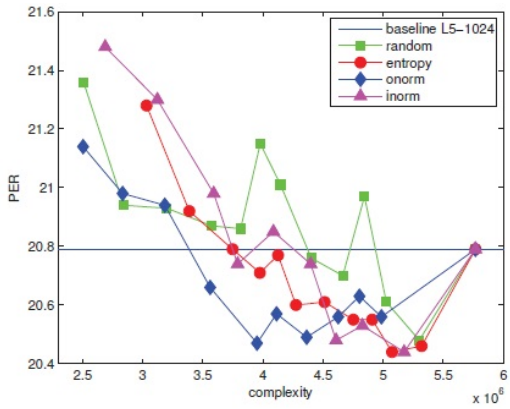

Figure 5.3 Performance of different node importance functions after re-finetuning on TIMIT

pared to the baseline DNN after retraining, and *o-norm* still has the best performance. In contrast, random importance function generally performed worse than the proposed ones and showed instability in performance change. This demonstrates that the proposed ones give better and more stable indications about the importance of hidden-nodes. From the above results, node pruning can effectively obtain 37.50% complexity reduction (from 5.6M to 3.5M) without hurting the decoding performance. Since *o-norm* showed the best and most stable performance, it was used for the following experiments.

5.3.2 Experiments on Switchboard

In this section node-pruning is evaluated on a LVCSR task, Switchboard English. For fast experiments, a subset consisting of data of 810 speakers(approximately 50 hours) was first selected from the whole 309 hours data set for training. 13-dimensional PLP features with per-speaker CMN and CVN, along with first and second derivatives were extracted. Cross-word triphone models with 3001 tied-states was used.

State alignment for DNN training was generated using a GMM model. A trigram language model which was trained on the transcription of the 2000h Fisher corpus and interpolated with a background trigram model was used for decoding. For all the experiments, the switchboard (`swb`) part of the Hub5'00 data set and the fisher (`fsh`) part of the RT03S data set were used as the test set in this paper, which is the same as ([33],

Seide et al., 2011: 24-29.).

The baseline DNN has 7 hidden layers of 2048 nodes per layer. Node pruning with *o-norm* importance function was applied and the results of the subset and full training set are shown in table 5.1.

System	#Pruned Nodes	DNN Complexity	WER (%)	
			swb	fsh
Baseline	—	32.2M	25.7	28.8
Node-Pruning	6000	14M	25.4	28.6
	6500	13.1M	25.6	28.9
	7000	**12.2M**	**25.5**	**28.8**
	7500	11.3M	25.9	29.0
	8000	10.4M	26.0	29.1

Table 5.1 Word error rate(WER) versus pruned complexity on switchboard 50 hours data

It can be seen that with 7000 hidden nodes pruned, about 62.1% DNN complexity reduction is obtained on the 50-hour task without any performance loss, which again demonstrates the effectiveness of the node-pruning approach.

We find that much model complexity can be pruned using node-pruning, it is then interesting to investigate the redundancy of model trained with different amount of data. So node-pruning is also conducted on the whole 300 hours data-set. Cross-word triphone models with 9296 tied-states was used, and other settings including hidden layer topology is the same as the one used in 50 hour experiments. Still, different number of hidden neurons are pruned from the DNN and the pruned model is then used for decoding on the test sets, experimental results are shown in table 5.2.

System	#Pruned Nodes	DNN Complexity	WER (%)	
			swb	fsh
Baseline	—	45M	19.7	21.7
Node-Pruning	6000	19.5M	19.8	22.0
	5000	**22.5M**	**19.6**	**21.7**

Table 5.2 Word error rate(WER) versus pruned complexity on switchboard 300 hours data

Because training on 300 hour data is very time-consuming, I'm not able to provide

very exhaustive experimental results here. The model on 300 hour data can afford to lose 50% complexity, the percentage is smaller than the model on 50 hour data(we need to mention that the output layer size in 300 hour model is much larger than the other one). This is consistent with the intuition that more data needs a more parameters to model.

5.3.3 Combination with SVD

As stated in section 4.2, node-pruning is complementary to weight matrices decomposition methods and may be combined to get further complexity saving. This is investigated on the 50 hours switchboard English task. Here, SVD decomposition is applied on the weight matrices of a node-pruned DNN. For comparison, standard SVD restructuring with different ranks was also performed. Note that SVD was applied on all matrices except the one above the input layer, because the input layer is relatively small(the dimension is 429).

SVD Rank	#Pruned Nodes	DNN Complexity	WER (%)	
			swb	fsh
192	—	6.5M	25.3	28.3
128	—	4.6M	25.9	29.0
192	**6500**	**3.95M**	**25.7**	**28.5**

Table 5.3 WER versus pruned SVD complexity on switchboard 50 hours data

Table 5.3 shows that DNN complexity can be effectively reduced to 20.1% (rank 192) by pure SVD restructuring. SVD with smaller rank will further reduce complexity at a cost of slight WER increase. By combining SVD rank-192 with node-pruning[1], a more compact DNN (87.7% complexity reduction) can be obtained. It is worth noting that the resultant compact DNN is smaller than SVD rank-128 and suffer no accuracy loss from baseline DNN(table 5.1), which means node-pruning combined with SVD can achieve better tradeoff between complexity and performance.

[1]SVD is only applied on matrices that has fairly large size (a r-rank SVD can reduce complexity of a $m \times n$ matrix only if $m \times r + r \times n < m \times n$)

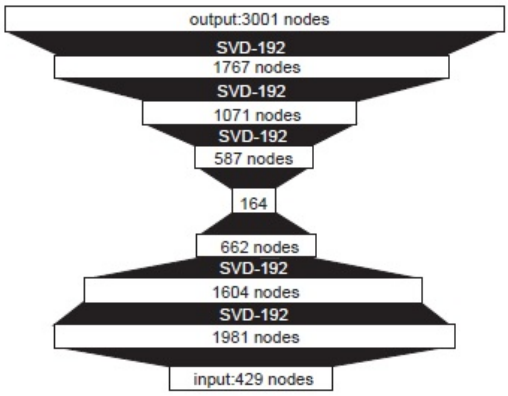

Figure 5.4 Illustration of node-pruning combined with SVD

5.3.4 The bottleneck shape

One interesting observation is that, in all the node-pruning experiments(especially on SWB task), without explicitly restraining the layer size, the reshaped DNN reveals an extremely small middle hidden layer as shown in figure 5.4. Bottleneck features from DNN is of great research interest([6], Yu and Seltzer, 2011: 237-240.),([7], Sainath et al., 2012: 4153-4156.), in those works, the middle hidden layer of DNN is **set** to be of a relatively small dimension, and activations from this layer is fed into a GMM-HMM system. However, for node-pruning we find that hidden neurons in the middle will usually get lower importance score. This bottleneck shape not only implies some rationality of using bottleneck features, but also may reveal new properties of DNN. The relevant investigation will be a future work. We show the shape of pruned DNN on various tasks in table 5.4.

metric & data set	shape of pruned DNN
onorm(TIMIT)	1323-990-737-578-793-1022-183
onorm(SWB50)	429-1860-1581-511-97-477-1071-1739-3001
onorm(SWB300)	429-1361-1997-1640-331-595-1340-1072-9296

Table 5.4 Shape of pruned DNN(including input and output layer)

5.4 Node-pruning for fast training

Since we use RBM-pretraining, it is interesting to apply node-pruning just after RBM-pretraining. The data set is 50 hour switchboard, we tried to prune 5500 hidden neurons on the RBM-initialized DNN, the resulting shape is 429-1843-1484-1015-922-950-574-2048-3001, the final hidden layer is not pruned because $o-norm$ is used and the final transformation matrix is not RBM-pretrained(random). Note that the number of hidden neurons is descending if we count from the input layer. Then, node-pruning is applied after only two iterations(the training usually needs more than ten iterations), we also prune 5500 hidden neurons, and the shape becomes 429-2005-1582-860-479-793-1069-2048-3001. It can be observed that the shape is like the one after a complete training process. This encourages us to do node-pruning after the first iteration and train the pruned DNN because smaller model complexity will speed up training, we call this approach *Lightly-discriminative pre-training* because it is the traditional pre-training plus a little fine-tuning. The drawback of this approach is that although we have proved that DNN usually has certain amount of node-level redundancy, it is hard to decide how many nodes can be pruned so exhaustive experiments as in section 5.3.2 are needed. In this work we propose to prune a moderate percentage of importance, as described in algorithm 4, because we believe that a moderate percentage will generalize well across different models and data-sets. Experimental results are shown in table 5.5, we need to mention that in this set of experiments ASGD([34], Zhang et al., 2013: 6660-6663.) is used for fast experiments.

Algorithm 4 Global Node Pruning by percentage

1: Set a percentage threshold e_{node}
2: Calculate importance score for all hidden nodes and sort them in an ascending order
3: Calculate the sum of all scores $SumScore$
4: Set $PrunedSums \leftarrow 0$ and $i \leftarrow 1$
5: **repeat**
6: Prune the node with the i^{th} smallest norm value from the sorted list
7: $PrunedSums \leftarrow PrunedSums + score_i$
8: $i \leftarrow i + 1$
9: **until** $\frac{PrunedSums}{SumScore} \geq e_{node}$

Table 5.5 Word error rate (WER) comparisons between node-pruned DNNs and standard DNNs. ASGD was used.

Pre-Training	e_{node}	Time	Complexity	WER (%)	
				swb	fsh
RBM	—	430m	32.2M	26.1	29.2
Lightly DT	0.10	190m	15.7M	25.9	28.9
	0.15	180m	14.2M	25.7	29.0
	0.20	**160m**	**12.6M**	**26.1**	**29.1**
	0.25	150m	11.0M	26.2	29.4

5.5 Summary

In this chapter, we report experimental results about node-pruning for DNN. First, to investigate the efficiency and nature of node-pruning, testing experiments are conducted on a relatively small data set TIMIT, it is found that all three proposed importance function worked well and random pruning will give unstable results. The *o-norm* importance function is chosen to be applied on a LVCSR task switchboard English. As an extension, results about node-pruning combined with SVD restructuring are reported, showing that the model complexity can be astonishingly reduced to 12.3%. As a side product, the bottleneck shape of the pruned DNN is also very interesting. Finally the possibility of using node-pruning for DNN training speed-up is discussed.

Chapter 6 Conclusion

The GMM-HMM framework has been the state-of-art system for ASR since the 1970s. It was not until recently that researchers were able to introduce DNN as part of the acoustic model, utilizing the powerful modelling power of DNN. This thesis mainly focuses on structure optimization about DNN, many recent works use different restructuring techniques that can reduce model complexity without performance loss. In this work, node-pruning is proposed to restructure DNN by removing redundant nodes according to some importance function. Experiments showed that Node-pruning can be used both for fast decoding or training speed-up.

6.1 Main contribution of this work

Despite the powerful modelling power of DNN, its vast number of parameters make it space-consuming to store in memory and time-consuming for decoding. Recently several work have been devoted to restructure DNN for model complexity reduction, which proved a sparseness constraint can be imposed on DNN training.

In this work, a new framework of reshaping DNN by node-pruning is proposed, which is complementary to the previously proposed approaches based on weight operation. Several novel node importance functions are proposed and experimented on the TIMIT task. The L1 norm of outgoing links, referred to as *onorm*, is shown to be most effective. With *onorm* node-pruning, on a 50 hours switchboard English task, the DNN complexity can be reduced to 37.9% without losing any performance. By further combination with SVD based weight matrices decomposition, up to 87.7% DNN complexity reduction can be achieved without affecting the recognition accuracy. The node pruning also reveals natural bottle-neck shape of DNN. We also showed node-pruning can be used for fast training by applying it on the DNN after one pass of training.

6.2 Future work

In this thesis some intuitional importance functions are proposed. However, the proposed importance function lack theoretical explanation of their effectiveness. Also, there are no proper guidance of how many hidden neurons are redundant given a neural network.

The point of interest is the bottleneck-shape of the node-pruned DNN. An interesting question is that since by the node-pruning framework we can get a bottleneck-shape DNN to work, we should also be able to train a DNN of bottleneck-shape in the first place. I tried training DNN with a bottleneck-shape but the trained model is not as good as baseline, the problem might be bad initialization. Another is the link to the bottleneck features, in works about bottleneck features the middle hidden layer is set to be small and the DNN trained in this way usually doesn't have baseline recognition performance. Since the node-pruned DNN still retains baseline performance, it can be used in the bottleneck feature framework.

REFERENCE

[1] RAHMAN MOHAMED A, DAHL G E, HINTON G E. Acoustic Modeling Using Deep Belief Networks[J]. IEEE Transactions on Audio, Speech & Language Processing, 2012, 20(1):14–22.

[2] DAHL G E, YU D, DENG L, et al. Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition[J]. IEEE Transactions on Audio, Speech, & Language Processing, 2012, 20:30–42.

[3] BENGIO Y, LAMBLIN P, POPOVICI D, et al. Greedy layer-wise training of deep networks[M]//Advances in Neural Information Processing Systems 19. Cambridge, MA: MIT Press, 2007:153–160.

[4] RAHMAN MOHAMED A, YU D, DENG L. Investigation of full-sequence training of deep belief networks for speech recognition.[C]//. KOBAYASHI T, HIROSE K, NAKAMURA S. INTERSPEECH.[S.l.]: ISCA, 2010:2846–2849.

[5] VESELY K, GHOSHAL A, BURGET L, et al. Sequence-discriminative training of deep neural networks.[C]//INTERSPEECH.[S.l.]: ISCA, 2013:2345–2349.

[6] YU D, SELTZER M L. Improved Bottleneck Features Using Pretrained Deep Neural Networks.[C]//INTERSPEECH.[S.l.]: ISCA, 2011:237–240.

[7] SAINATH T N, KINGSBURY B, RAMABHADRAN B. Auto-encoder bottleneck features using deep belief networks.[C]//ICASSP.[S.l.]: IEEE, 2012:4153–4156.

[8] VANHOUCKE V, SENIOR A, MAO M Z. Improving the speed of neural networks on CPUs[C]//Proc. Deep Learning and Unsupervised Feature Learning Workshop, NIPS. .[S.l.]: [s.n.] , 2011.

[9] YU D, SEIDE F, LI G, et al. Exploiting sparseness in deep neural networks for large vocabulary speech recognition.[C]//ICASSP.[S.l.]: IEEE, 2012:4409–4412.

[10] XUE J, LI J, GONG Y. Restructuring of deep neural network acoustic models with singular value decomposition.[C]//.[S.l.]: ISCA, 2013:2365–2369.

[11] SAINATH T N, KINGSBURY B, SINDHWANI V, et al. Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets.[C]//ICASSP.[S.l.]: IEEE, 2013:6655–6659.

[12] TIANXING H, YUCHEN F, QIAN Y, et al. Reshaping Deep Neural Network for Fast Decoding by Node-pruning[C]//In proceedings of International Conference on Acoustics, Speech and Signal Processing(ICASSP). Florence, Italy: [s.n.] , 2014:245–249.

[13] REED R. Pruning algorithms-a survey[J]. IEEE transactions on Neural Networks, 1993, 4(5):740–747.

[14] SEGEE B E, JCARTER M. Fault tolerance of pruned multilayer networks[J]. Neural Networks, 1991, 2:447–452.

[15] YAMAMOTO S, TOSHINO, TMORI, et al. Gradual reduction of hidden units in the back propagation algorithm, and its application to blood cell classification[C]//Proc. International Joint Conference on Neural Networks. .[S.l.]: [s.n.] , 1993.

[16] DAVIS S B, MERMELSTEIN P. Comparison of parametric representations for monosyllabic word recognition in continuous spoken sentences[J]. IEEE Transactions on Acoustics, Speech and Signal Processing, 1980, 28:357–366.

[17] HHERMANSKY. Perceptual linear prediction of speech[J]. Journal of the acoustic society of America, 1990, 87(4):1738–1752.

[18] RABINER L, JUANG B H. Fundamentals of Speech Recognition[M].[S.l.]: Prentice Hall PTR, 1993.

[19] YU K. Adaptive Training for Large Vocabulary Continuous Speech Recognition[].

[20] CHEN S F, GOODMAN J. An empirical study of smoothing techniques for language modeling[C]//Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL-1996). .[S.l.]: [s.n.] , 1996:310–318.

[21] HERMANSKY H, ELLIS D P W, SHARMA S. Tandem connectionist feature extraction for conventional HMM systems[C]//PROC. ICASSP. .[S.l.]: [s.n.] , 2000:1635–1638.

[22] HINTON G E, OSINDERO S. A fast learning algorithm for deep belief nets[J]. Neural Computation, 2006, 18:1527–1554.

[23] HINTON G E, DENG L, YU D, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups[J]. Signal Processing Magazine, IEEE, 2012, 29(6):82–97.

[24] CYBENKO G. Approximation by superpositions of a sigmoidal function[J]. Mathematics of Control, Signals, and Systems (MCSS), 1989, 2(4):303–314.

[25] HORNIK K, STINCHCOMBE M, WHITE H. Multilayer feedforward networks are universal approximators[J]. Neural Networks, 1989, 2(5):359–366.

[26] ABDEL-HAMID O, DENG L, YU D. Exploring convolutional neural network structures and optimization techniques for speech recognition.[C]//INTERSPEECH.[S.l.]: ISCA, 2013:3366–3370.

[27] DAHL G E, SAINATH T N, HINTON G E. Improving deep neural networks for LVCSR using rectified linear units and dropout.[C]//ICASSP.[S.l.]: IEEE, 2013:8609–8613.

[28] HINTON G E. Training Products of Experts by Minimizing Contrastive Divergence[J]. Neural Computation, 2002, 14:1771–1800.

[29] FISCHER A, IGEL C. 2012. An Introduction to Restricted Boltzmann Machines.[C]//CIARP.[S.l.]: Springer, Lecture Notes in Computer Science, vol. 7441.

[30] SIETSMA J, DOW R J F. Creating artificial networks that generalize[J]. Neural Networks, 1991, 4:67–69.

[31] YOUNG S J, KERSHAW D, ODELL J, et al. The HTK Book Version 3.4[M].[S.l.]: Cambridge University Press, 2006.

[32] VINYALS O, MORGAN N. Deep vs. wide: depth on a budget for robust speech recognition.[C]//INTERSPEECH.[S.l.]: ISCA, 2013:114–118.

[33] SEIDE F, LI G, CHEN X, et al. Feature engineering in Context-Dependent Deep Neural Networks for conversational speech transcription.[C]//. NAHAMOO D, PICHENY M. ASRU.[S.l.]: IEEE, 2011:24–29.

[34] ZHANG S, ZHANG C, YOU Z, et al. Asynchronous stochastic gradient descent for DNN training.[C]//ICASSP.[S.l.]: IEEE, 2013:6660–6663.

Appendix A Publications during the undergraduate

- **Tianxing He**, Yuchen Fan, Yanmin Qian, Tian Tan, and Kai Yu. Reshaping Deep Neural Network for Fast Decoding by Node-pruning. Accepted by International Conference on Acoustics, Speech and Signal Processing(ICASSP), 245-249, Florence, Italty, 2014
- Yanmin Qian, **Tianxing He**, Wei Deng, Kai Yu. Automatic Node Pruning for Fast Back-Propagation for Deep Neural Network. Submitted to IEEE SLT 2014

Acknowledgements

Great thanks to my supervisor Yanmin Qian, who guided my experiments and writing for this thesis, and my mentor Kai Yu in the speech lab, who lead me into the world of speech recognition. I need to thank my friends in the SJTU speech lab, Yuchen Fan taught me how to run experiments and had some valuable discussions with me, Tian Tan taught me how to run baseline experiments for both GMM-HMM and DNN-HMM asr system. Of course I need to thanks authors of HTK and TNET that enable us to efficiently perform speech recognition experiments. I also need to thank Wei Deng for his code of ASGD, which trains DNN really fast.